

版本信息:

版本:REV2020

时间:30/01/2020

# 米联客 2020 版 FPGA 课程(MIG DDR 篇)

Milianke(msxbo) 2020 FPGA Course

电子版自学资料配视频课程

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  
版权所有

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

微信扫码注册米联客技术论坛 [www.uisrc.com](http://www.uisrc.com) 免费享受更多资源



扫码关注微信公众平台“MSXBO(米联客)”掌握更多信息动态



版本	时间	描述
Rev2016	2016-07-03	第一版
Rev2019	2019-05-09	第二版
Rev2020	2020-05-20	第三版

序:

FPGA 芯片是硬件技术而 FPGA 编程又称为硬件编程语言和流行的各类软件编程语言 C/C++、JAVA、python 等相比,掌握基础的硬件编程语言不是难事,难点是 FPGA 在每个专业领域的应用,只有充分理解了 FPGA,并且具有对自己所处行业专业背景认知,才能真正理解 FPGA 应该用在什么场合更加合适。

从业多年来,亲身经历了 FPGA 的发展历程,也深刻体会到未来 FPGA 应用领域可能发生的深刻变革,FPGA 从简单的逻辑门,发展到现在具备很多高速通信接口,而且最新 SOC 中也集成了 FPGA 单元,实现了 ARM 和 FPGA 单芯片。目前 XILINX 代表了业内领先的 FPGA 技术,已经可以把 FPGA\ARM\GPU\RFDID 等集成到单芯片。FPGA 的逻辑资源也是达到了前所未有的密度以及超大容量。

很多人问我学习 FPGA 是否有前途,这个问题着实难以回答。我们可以一起来探讨以下几种情况,会是 FPGA 发挥作用的场合:

#### 1)、数字 IC 设计工作

数字 IC 设计还是主要以硬件编程语言去设计数字 IC 芯片,通过硬件编程语言,软件可以把语言翻译成电路,自动布线工具可以完成布局布线,之后流片。由于流片费用非常贵,前期也可以用 FPGA 芯片模拟设计的数字 IC 芯片的功能。

#### 2)、高速模数字信号采集分析

高速的 ADC,DAC 的数模信号处理的领域也是必然需要用到 FPGA,在无线通信、雷达信号处理等领域也都会用到。

#### 3)、数字信号高速通信

FPGA 具备的高速接口也非常适合用于高速通信,比如 PCIE 通信、光通信、以太网通信

#### 4)、视频图像

包括图像的拼接、缩放、高效的实时传输等, 4K 视频、8K 视频领域

#### 5)、硬件加速算法

FPGA 的硬件加速领域也是目前的热门研究,也是 FPGA 未来最有前景的一个应用领域,已经有很多公司利用 FPGA 的硬件加速实现了很好的经济效应,但是目前 FPGA 的加速还没有做到普及,和传统的 GPU 相比,主要难度还是在开发难度上,一般小公司很难有实力取得突破。

#### 6)、通用 CPU GPU 无法完成的工作

如果通用的 CPU 和 GPU 无法完成的工作任务,可以考虑下 FPGA 或者带 FPGA 的 SOC.

FPGA 到目前为止依然是一个小众的领域,如果专门为了学习 FPGA 而学习 FPGA 而不知道如何应用 FPGA,那么这是非常悲哀的一件事情,学习 FPGA 只是学习一门技能,而结合自己专业背景选择最合适的解决方案,解决问题才是最终的目的。

米联客团队励志在 FPGA 领域可以贡献一份自己的力量，我们的目标是，可以减轻 FPGA 从业者基础学习难度、FPGA 应用难度，为中小型 FPGA 团队提供必要有价值的技术资料 and 硬件支持。我们希望和广大客户形成紧密的合作伙伴关系，一起创造共赢，各自实现自己的价值目标。

2020 年注定是不平凡的一年，是米联客团队继续成长的一年，也是米联客发展非常关键的一年，其中 2020 版本教程的研发也是新一年中米联客最重要的核心工作之一。

2020 版本教程是适应当前 FPGA 技术发展，SOC 技术发展，新形势下，米联客做出的战略决策。2020 版本教程需要解决以下几个问题：

- 1)、FPGA 基础课程，需要解决长期以来没有认真解决好的课程内容，包括：FPGA 的构架、FPGA 常用 IP 使用、硬件编程经验、通信接口应用、代码规范、时序分析
- 2)、新版本 vitis 软件的使用技巧
- 3)、更多更详细的讲解高速通信的基础知识和更多的应用解决方案
- 4)、对于 SOC (ZYNQ 和 MPSOC)Linux 课程做到适合初级入门，并提供丰富的应用 demo
- 5)、适应未来发展趋势，增加 FPGA 高层次加速算法编程领域的课程内容
- 6)、教程的构架能够支持 7 系列 FPGA、UltraScale 系列 FPGA、UltraScale+系列 FPGA 和 MPSOC

不管有多麻烦，不管困难多大，面对挑战我们坚信胜利！

米联客团队  
2020 年 1 月 30 日

## 目录

米联客 2020 版 FPGA 课程(MIG DDR 篇).....	1
01 使用 MIG 对 DDR 测试详解.....	5
1.1MIG 控制器概述 .....	5
1.2 用户 FPGA 逻辑接口 .....	5
1.3 时钟要求 .....	6
1.4MIG 内存控制器用户逻辑时序 .....	6
1.4.1 控制命令 .....	7
1.4.2 写数据时序 .....	7
1.4.3 读数据时序 .....	8
1.5 搭建测试工程 .....	9
1.5.1 添加 MIG IP CORE .....	9
1.5.2 添加时钟 PLL .....	17
1.5.3 编写测试代码 .....	19
1.5.4 搭建完成 .....	24
1.6 在线仿真 .....	24
1.6.1 添加仿真信号 .....	24
1.6.2 在线仿真结果 .....	26
1.7 软件仿真 .....	27
1.7.1 用自带 demo 产生测试工程 .....	27
1.7.2 添加仿真文件并测试 .....	28
1.7.3 修改必要的仿真文件参数 .....	28
1.7.4 仿真结果 .....	39
1.8 VIVADO2019MIG 向导的 Bug.....	41
02 基于 MIG 实现三缓存构架设计 .....	42
2.1 概述 .....	42
2.2 构架设计 .....	42
2.3edma_ctr.v .....	43
2.3.1ms_fifo ip 设置 .....	49
2.3.2wr_fifo ip 设置 .....	50
2.3.3rd_fifo ip 设置 .....	52
2.3.4 帧信号写入消息 FIFO.....	53
2.3.5M_S 内存管理状态机 .....	54
2.3.6 读写 DDR 请求控制 .....	56
2.4 代码结构 .....	57
2.5RTL 仿真结果 .....	57
2.6 输出测试图形 .....	58
03 摄像头采集 DDR 三缓存 .....	59
3.1 概述 .....	59
3.2 构架设计 .....	59
3.3 代码结构 .....	60
3.4 安装摄像头 .....	60
3.5 测试结果 .....	61
3.5.1OV7725 .....	61
3.5.2OV5640 .....	61
3.5.3MT9V034 .....	62
04HDMI 视频输入 DDR 三缓存 .....	63
4.1 概述 .....	63

---

4.2 构架设计 .....	63
4.3 代码结构 .....	64
4.4HDMI 接线 .....	64
4.5 测试结果 .....	65

# 01 使用 MIG 对 DDR 测试详解

软件版本: VIVADO2019.2

操作系统: WIN10 64bit

硬件平台: 适用 XILINX A7/K7/Z7/ZU/KU 系列 FPGA

登录米联客(MSXBO)FPGA 社区-[www.uisrc.com](http://www.uisrc.com) 观看免费视频课程、在线答疑解惑!

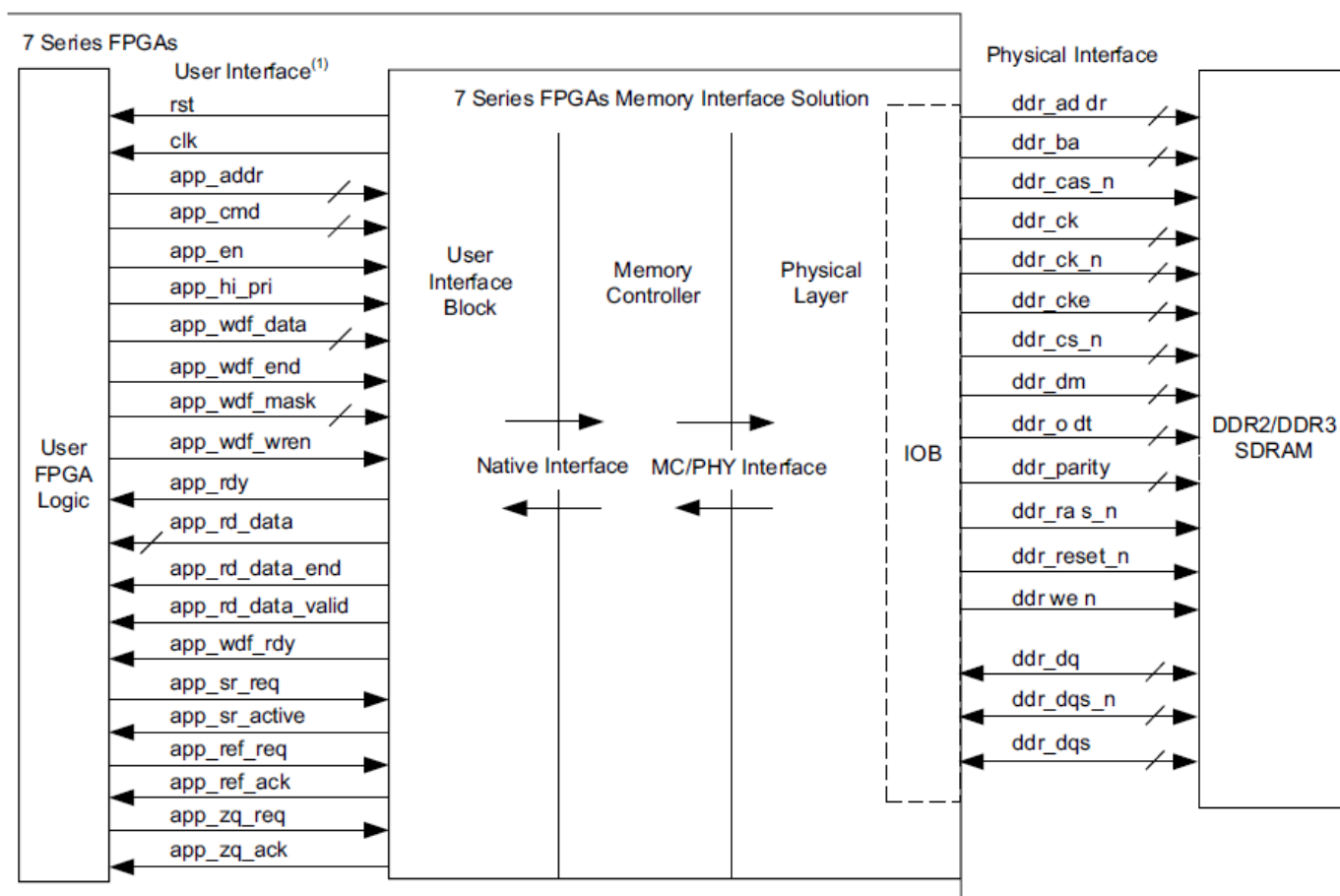
## 1.1MIG 控制器概述

如下图所示, MIG 控制器 IP 提供了 User FPGA Logic 方便用户实现对 DDR 的访问, 而不需要让用户再花费太多精力编写 DDR 控制接口。MIG 控制器 IP 包括了 3 大结构:

- 1)、User Interface Block(用户控制逻辑接口)
- 2)、Memory Controller(内存控制器)
- 3)、Physical Layer(物理层接到 DDR 芯片)

MIG 控制器不仅仅支持下图中采用的简单用户控制逻辑接口, 也可以支持 AXI4 总线接口。本文讲解简单用户控制逻辑接口的访问方式。关于标准 AXI4 总线方式应用方案, 另外一篇关于米联客编写的 FDMA IP 应用的方案中有讲解。标准 AXI4 总线方式相比本文的方式, 通用性更好。

关于 MIG 控制的描述可以阅读官方文档 ug586。



1. System clock (sys\_clk\_p and sys\_clk\_n/sys\_clk\_i), Reference clock (clk\_ref\_p and clk\_ref\_n/clk\_ref\_i), and system reset (sys\_rst\_n) port connections are not shown in block diagram.

Series FPGAs Memory Interface Solution

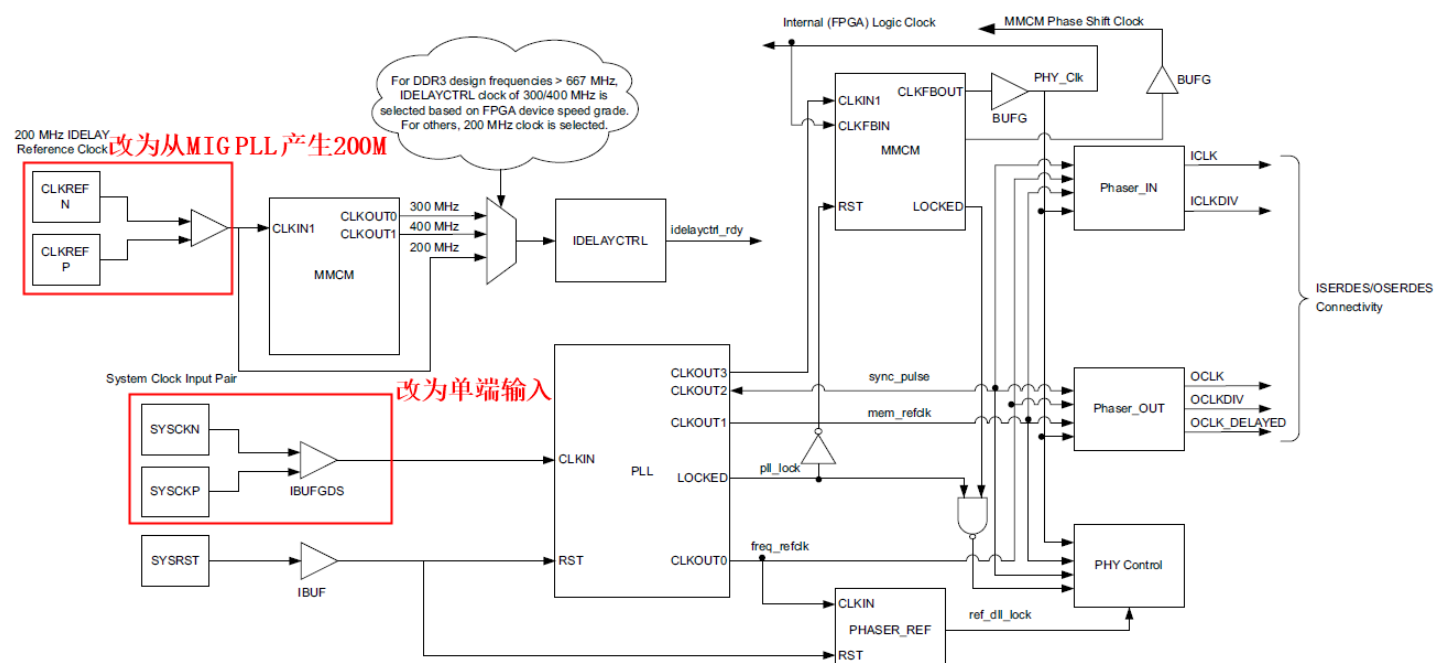
## 1.2 用户 FPGA 逻辑接口

对于广大用户从应用角度, 只需要掌握 User Interface Block 用户接口的使用, 以及 Physical Layer 物理层的管脚

定义。所以我们先看看用户接口信号定义。后面默认我们用“用户接口”指代 User Interface Block。

信号	方向	描述
app_addr[ADDR_WIDTH - 1:0]	input	内存地址定义，可寻址的范围，地址位宽是 ADDR_WIDTH
app_cmd[2:0]	input	读写命令，001 读；000 写
app_en	input	控制命令使能
app_rdy	input	MIG 控制器控制通道准备好
app_hi_pri	output	提高当前请求的优先级，高电平有效，目前米联客的例子中没有使用这个信号
app_rd_data [APP_DATA_WIDTH - 1:0]	input	读数据通道
app_rd_data_end	output	读数据结束
app_rd_data_valid	output	读数据有效
app_sz	input	预留，设置为 0
app_wdf_data [APP_DATA_WIDTH - 1:0]	input	写数据通道
app_wdf_end	input	写数据结束
app_wdf_mask [APP_MASK_WIDTH - 1:0]	input	数据掩码
app_wdf_rdy	output	写准备好
app_wdf_wren	input	写使能
app_correct_en_i	input	ECC 使用的时候，设置高电平使能
app_sr_req	input	预留，设置为 0
app_sr_active	output	预留

## 1.3 时钟要求



7 代 FPGA MIG 时钟支持单端外部时钟输入，内部时钟输入，官方推荐的是采用 200M 差分时钟提供给 idelay 模块使用，系统时钟也采用差分时钟输入。米联客没有完全参考官方的设计方案，SYSCLK 设置为单端时钟输入，并且利用 MIG 自带的 PLL 产生 200M 参考时钟提供给 idelay 模块使用。

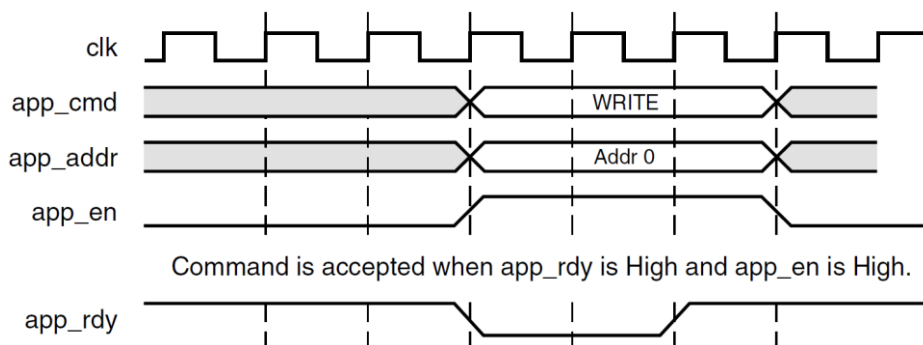
## 1.4 MIG 内存控制器用户逻辑时序

本小节讲解控制命令、读命令、写命令及其时序。



## 1.4.1 控制命令

当用户逻辑 app\_en 和 app\_rdy 同时有效，命令被写入控制命令 FIFO。

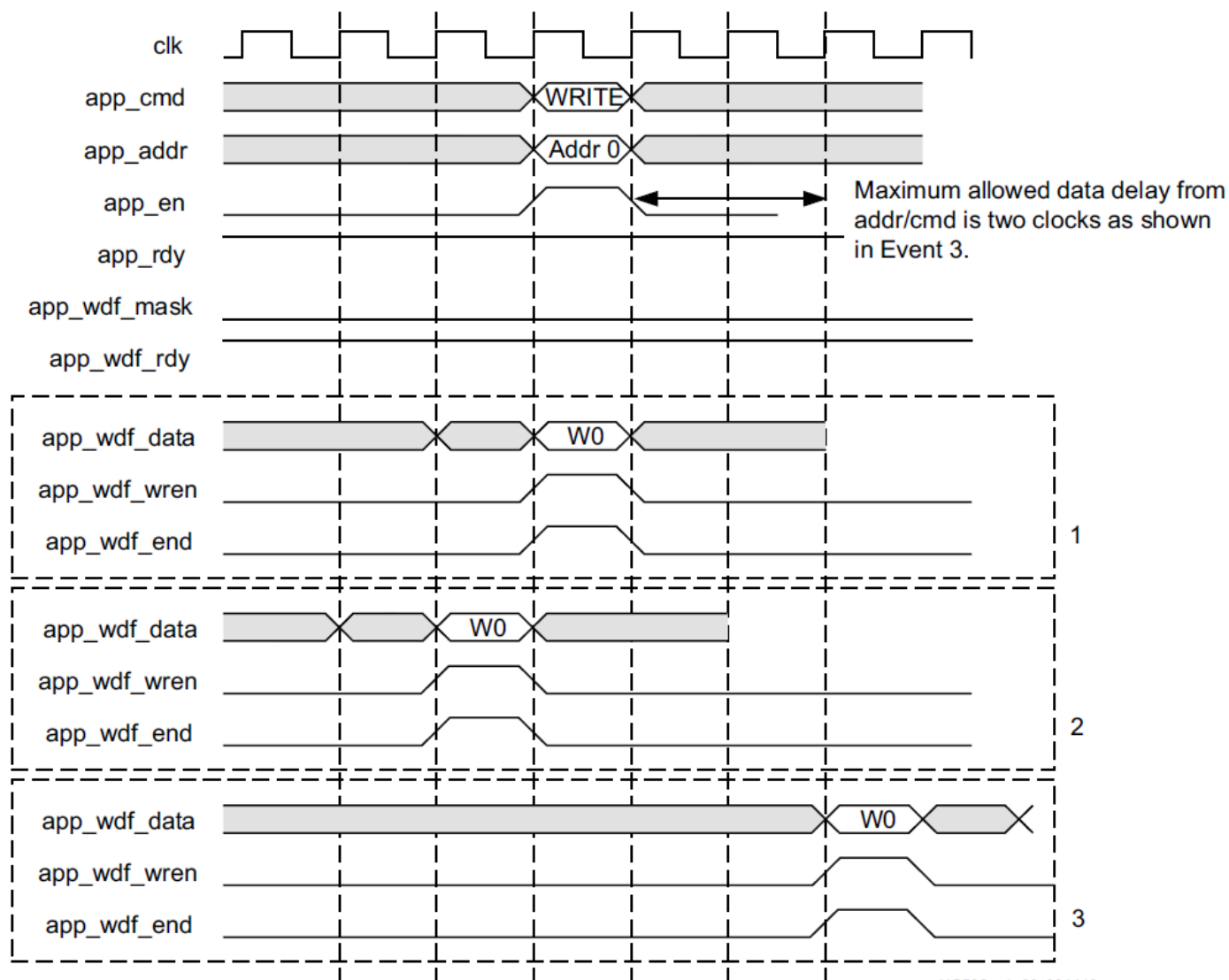


Command Timing Diagram with app\_rdy Asserted

## 1.4.2 写数据时序

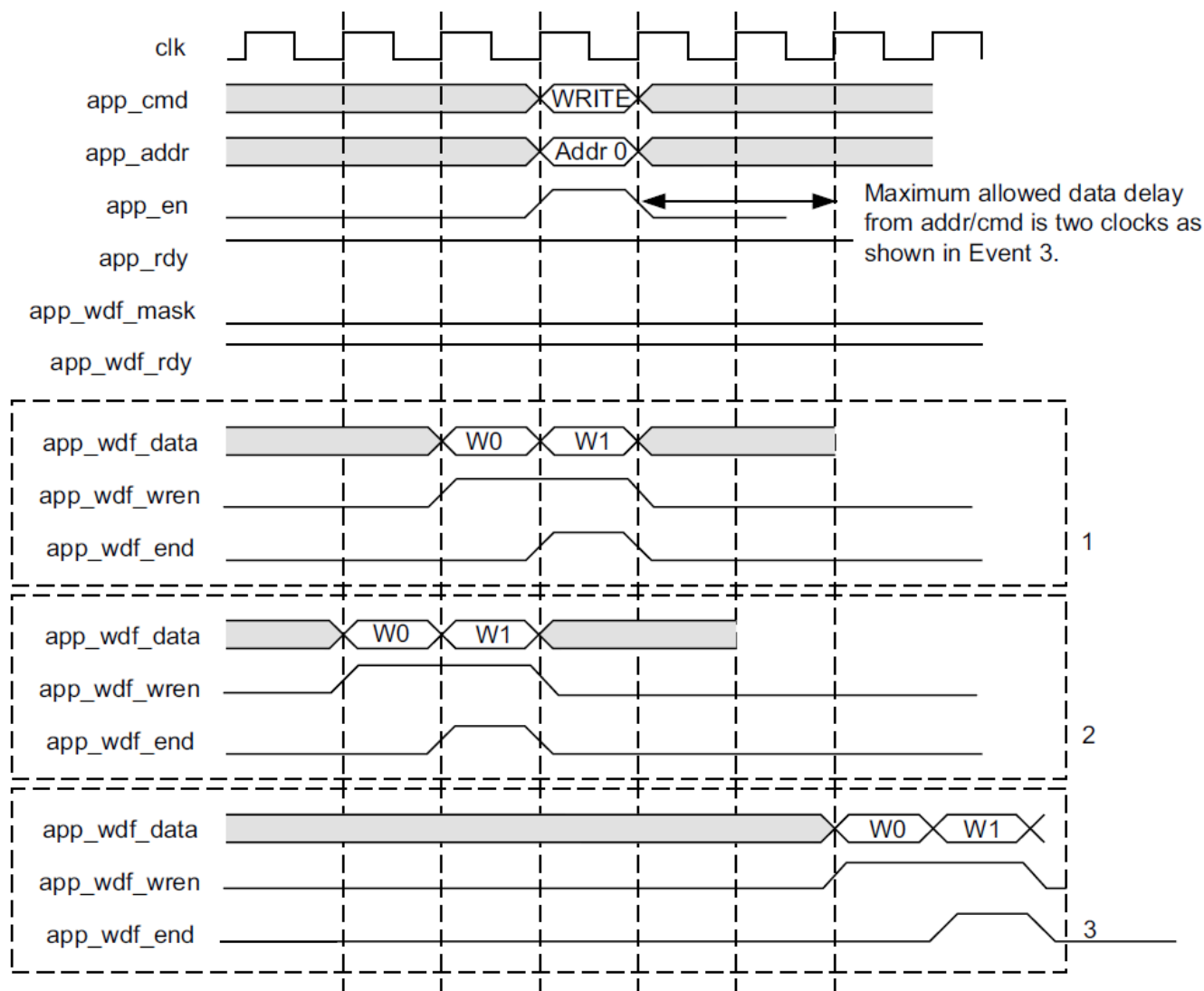
当 app\_wdf\_wren 和 app\_wdf\_rdy 同时有效，写数据被写入数据写 FIFO。app\_wdf\_mask 信号可用于屏蔽写入外部存储器的字节。

此外下图中说明控制命令和数据通道没有先后顺序要求，但是要求控制命令发送后，数据必须最大延迟不能超过 2 个时钟。



UG586 c1 63 091410

4:1 Mode UI Interface Write Timing Diagram (Memory Burst Type = BL8)

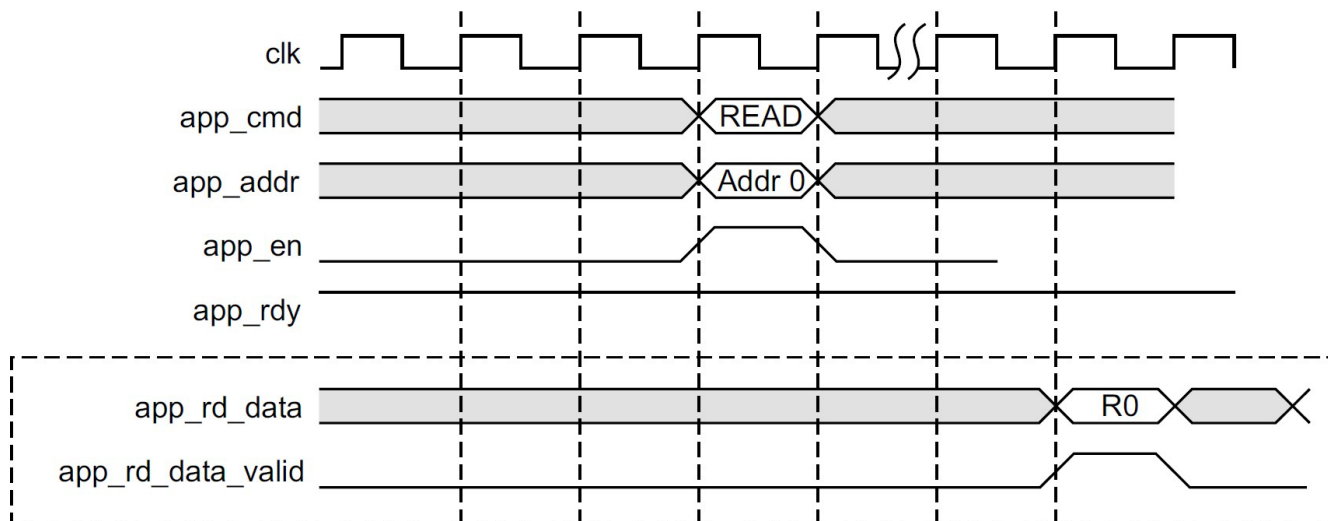


2:1 Mode UI Interface Write Timing Diagram (Memory Burst Type = BL8)

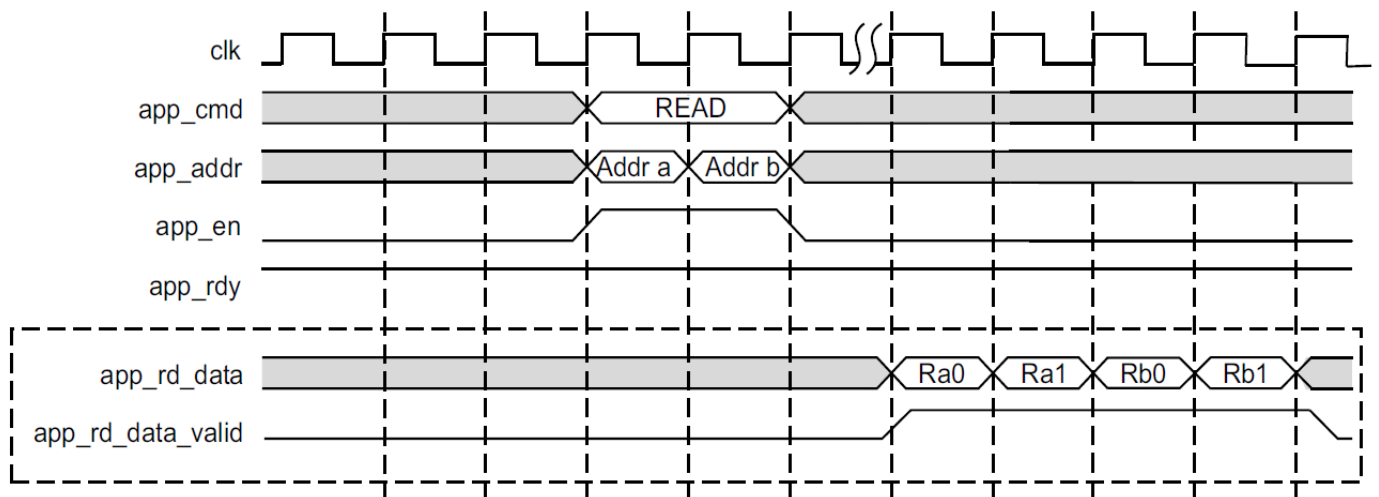
必须使用 **app\_wdf\_end** 信号来指示存储器写入突发的结束。对于 2:1 模式，应该在第二个写入数据字上断言 **app\_wdf\_end** 信号。对于 4:1 模式可以让 **app\_wdf\_end=app\_wdf\_wren** 简化时序控制。

### 1.4.3 读数据时序

读命令发送后，当 **app\_rd\_data\_valid** 有效数据有效。**app\_rd\_data\_end** 信号表示一次读数据结束。



4:1 Mode UI Interface Read Timing Diagram (Memory Burst Type = BL8)



2: 1Mode UI Interface Read Timing Diagram(Memory Burst Type = BL4 or BL8)

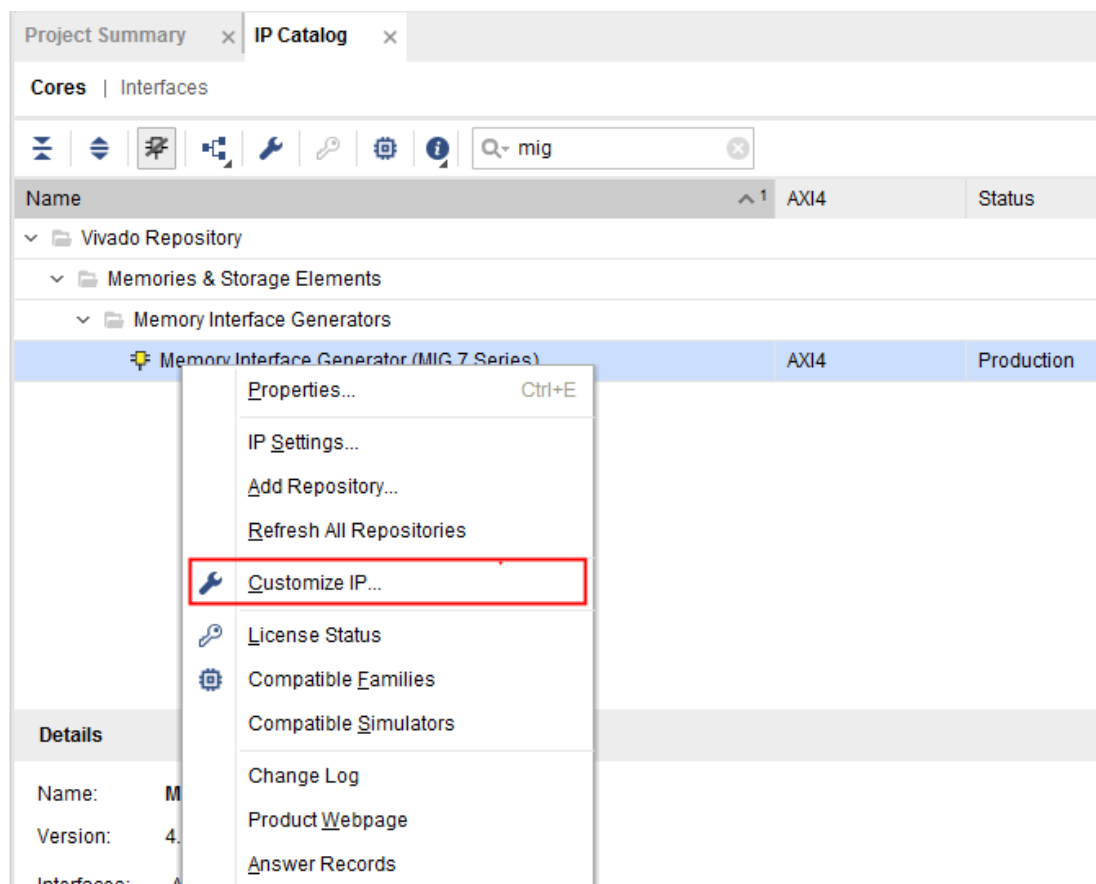
以上理论时序方面的信息我们做了大量精简,提取了米联客实际应用工程中有用的信息,掌握这些信息足够驾驭 MIG 的使用。如果读者需要更多了解 MIG 的其他信息,自行阅读官方文档 ug586

## 1.5 搭建测试工程

以下以 MA703FA-100T 为样板做说明,对于 MA703FA-35T DDR 只焊接了 1 片数据数据位宽选 16bit,如果有疑问打开配套代码查看对比各项参数。

### 1.5.1 添加 MIG IP CORE

任意创建一个新的空的工程(创建工程的具体工程如果还不清楚的看我们教程第一季部分)



单击 Next

**Memory Interface Generator**

The Memory Interface Generator (MIG) creates memory controllers for Xilinx FPGAs. MIG creates complete customized Verilog or VHDL RTL source code, pin-out and design constraints for the FPGA selected, and script files for implementation and simulation.

**Vivado Project Options**

This GUI includes all configurable options along with explanations to aid in generation of the required controller. Please note that some of the options selected in the Vivado Project Options will be used in generation of the controller. It is very important that the correct Vivado Project Options are selected. These options are listed below.

Selected Vivado Project Options:

Fpga Family : Artix-7

Fpga Part : xc7a100t-fgg484

Speed Grade : -2

Synthesis Tool : VIVADO

Design Entry : VERILOG

If any of these options are incorrect, please click on "Cancel", change the Vivado Project Options, and restart MIG. This version of MIG is tested with Vivado 2018.3 or later, it is not tested with previous versions of Vivado.

[User Guide](#)[Next >](#)[Cancel](#)

选择 Create Design 单击 Next

Memory Interface Generator

**MIG Output Options**☒ **Create Design**

Select this option to generate a memory controller. Generating a memory controller will create RTL, XDC, implementation and simulation files.

☐ **Verify Pin Changes and Update Design**

Selecting this feature verifies the modified XDC for a design already generated through MIG. This option will allow you to change the pin out and validate it instantly. It updates the input XDC file to be compatible with the current version of MIG. While updating the XDC it preserves the pin outs of the input XDC. This option will also generate the new design with the Component Name you selected in this page.

**Component Name**

Please specify the component name for the memory interface. The design directories will be generated under a directory with this name. Three directories will be created "example\_design", "user\_design" and "docs". The user\_design will contain the generated memory interface. The example\_design adds a simple example application connected to the generated memory interface.

Component Name

**Multi-Controller**

Up to maximum of 8 controllers with a combination of DDR3 SDRAM, QDR II+ SRAM or RLDRAM II can be generated. The number of controllers that can be accommodated may be limited by the data width and the number of banks available in device. Refer user guide for more information


Number of controllers

**AXI4 Interface**

Enables the AXI4 interface. AXI4 interface is supported only for DDR3 SDRAM and DDR2 SDRAM controllers with Verilog design entry.

☐ AXI4 Interface[User Guide](#)[< Back](#)[Next >](#)[Cancel](#)

继续 Next



**Pin Compatible FPGAs**

Pin Compatible FPGAs include all devices with the same package and speed grade as the target device. Different FPGA devices with the same package do not have the same bonded pins. By selecting Pin Compatible FPGAs, MIG will only select pins that are common between the target device and all selected devices. Use the default XDC in the par folder for the target part. If the target part is changed, use the appropriate XDC in the compatible\_uct folder. **If a Pin Compatible FPGA is not chosen now and later a different FPGA is used, the generated XDC may not work for the new device and a board spin may be required.** MIG only ensures that MIG generated pin out is compatible among the selected compatible FPGA devices. Unselected devices will not be considered for compatibility during the pin allocation process.

A blank list indicates that there are no compatible parts exist for the selected target part and this page can be skipped.

Note that different parts in the same package will have different internal package skew values. De-rate the minimum period appropriately in the Controller Options page when different parts in the same package are used. Consult the User Guide for more information.


Target FPGA:

**Pin Compatible FPGAs**

- ☐ artix7
  - ☐ 7a
    - ☐ xc7a35t-fgg484
    - ☐ xc7a50t-fgg484
    - ☐ xc7a75t-fgg484
    - ☐ xc7a15t-fgg484

[User Guide](#) [< Back](#) [Next >](#) [Cancel](#)

选择 DDR3 继续 Next



**Memory Selection**

Select the type of memory interface. Please refer to the User Guide for a detailed list of supported controllers for each FPGA family. The list below shows currently available interface(s) for the specific FPGA, speed grade and design entry chosen.

Select the controller type:



- ☒ DDR3 SDRAM
- ☐ DDR2 SDRAM
- ☐ LPDDR2 SDRAM

[User Guide](#) [< Back](#) [Next >](#) [Cancel](#)

设置最高运行频率、内存颗粒型号、内存的数据位宽，对于 A7 最高 400M 主频(800Mbps 数据速度)，下图种对于 MA703FA-35T Data Width 设置为 16



设置时钟输入类型如下设置



Pin Compatible FPGAs  
Memory Selection  
Controller Options  
AXI Parameter  
Memory Options  
**FPGA Options**  
Extended FPGA Options  
IO Planning Options  
Pin Selection  
System Signals Selection  
Summary  
Simulation Options  
PCB information  
Design Notes

**System Clock**  
Choose the desired input clock configuration. Design clock can be Differential or Single-Ended.  
System Clock: No Buffer

**Reference Clock**  
Choose the desired reference clock configuration. Reference clock can be Differential or Single-Ended.  
Reference Clock: Use System Clock

**System Reset Polarity**  
Choose the desired System Reset Polarity.  
System Reset Polarity: ACTIVE LOW

**Debug Signals Control**  
This feature allows various debug signals present in the IP to be monitored on the ChipScope tool. The debug signals include status signals of various PHY calibration stages. Enabling this feature will connect all the debug signals to the ChipScope ILA and VIO cores in the example design top module. A part of each bus in the debug interface has been grounded so that users can replace the grounded signals with the required signals.  
Debug Signals for Memory Controller: OFF

**Sample Data Depth**  
This selects the value of Sample Data depth for Chipscope ILA used in Debug logic.  
Sample Data Depth: 1024



**Internal Vref**  
Internal Vref can be used to allow the use of the Vref pins as normal IO pins. This option can only be used at 800 Mbps and lower data rates. This can free 2 pins per bank where inputs are used. This setting has no effect on banks with only outputs.  
Internal Vref: ☐

**IO Power Reduction**  
Significantly reduces average IO power by automatically disabling DQ/DQS IBUFs and internal terminations during WRITES and periods of inactivity.  
IO Power Reduction: ON

**XADC Instantiation**  
The memory interface uses the temperature reading from the XADC block to perform temperature compensation and keep the read DQS centered in the data window. There is one XADC block per device. If the XADC is not currently used anywhere in the design, enable this option to have the block instantiated. If the XADC is already used, disable this MIG option. The user is then required to provide the temperature value to the top level 12-bit device\_temp\_i input port. Refer to Answer Record 51687 or the UG586 for detailed information.  
XADC Instantiation: Enabled

User Guide < Back Next > Cancel

设置终端电阻只针对 HR BANK

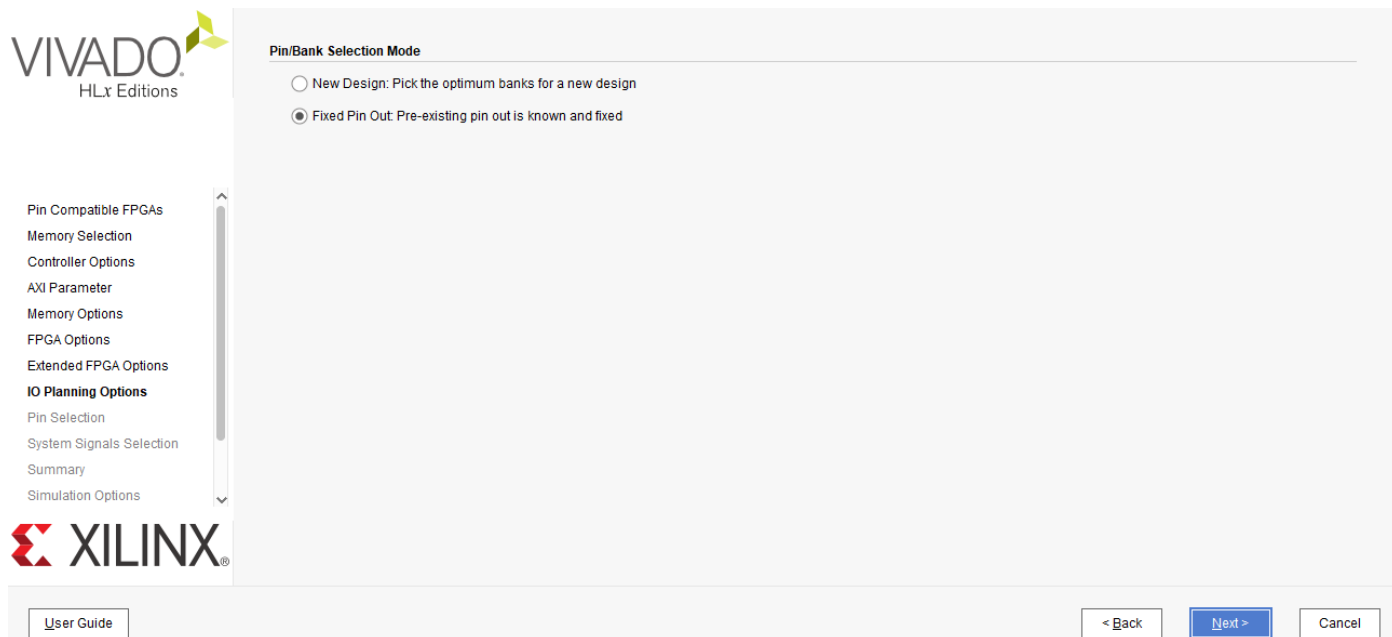


Pin Compatible FPGAs  
Memory Selection  
Controller Options  
AXI Parameter  
Memory Options  
FPGA Options  
**Extended FPGA Options**  
IO Planning Options  
Pin Selection  
System Signals Selection  
Summary  
Simulation Options

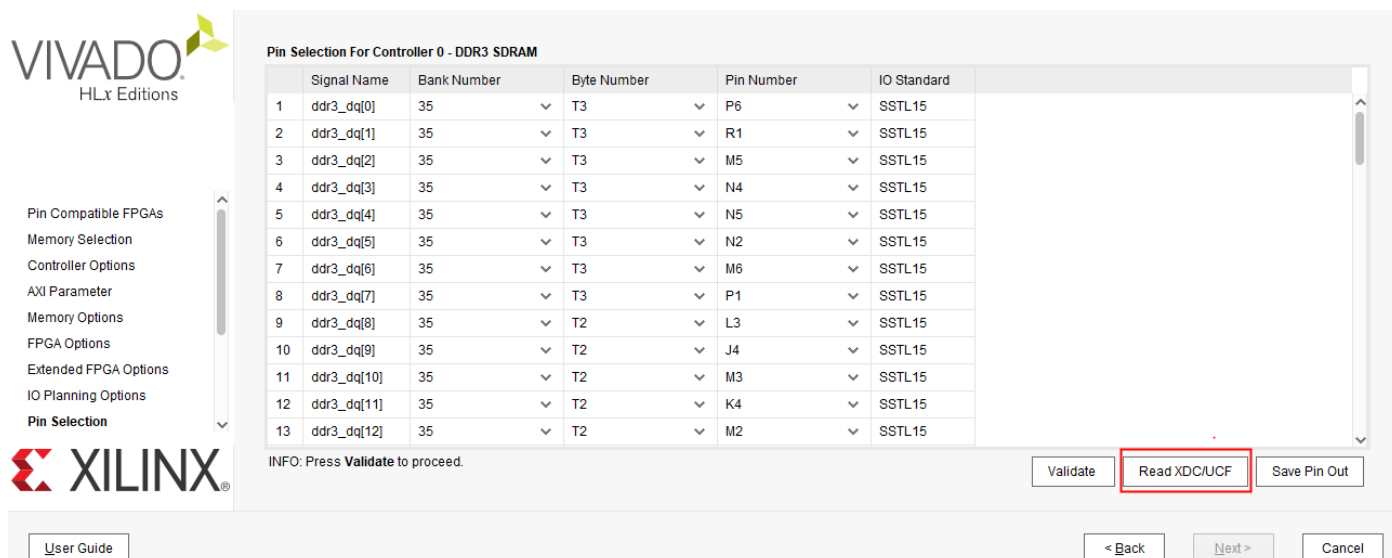
**Internal Termination for High Range Banks**  
Select the internal termination (IN\_TERM) impedance for the High Range (HR) banks. This setting applies **only** to the HR banks used in the interface.  
Internal Termination Impedance: 50 Ohms

User Guide < Back Next > Cancel

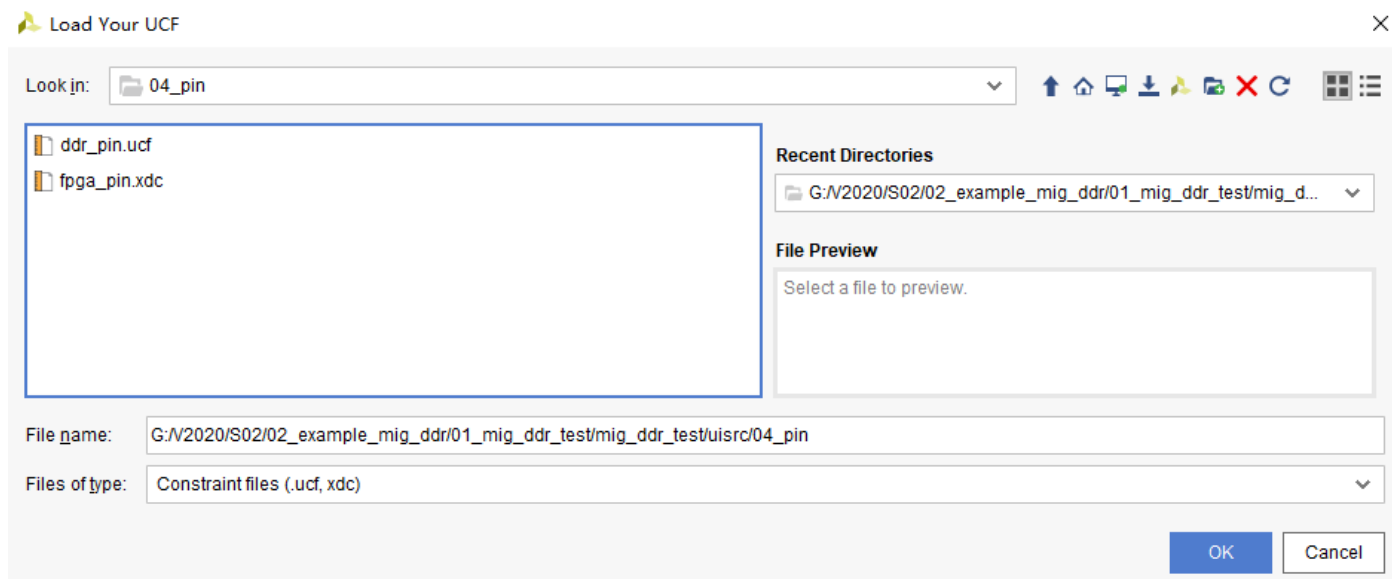
准备导入 DDR PIN 脚约束



读取../uisrc/pin 路径下的 ddr 管脚定义(如果是新设计的硬件需要根据原理图设置)



读取 ddr\_pin.ucf 文件



添加完成后需要校验 validate 后 Next



VIVADO<sup>®</sup>

HLx Editions

Pin Compatible FPGAs

Memory Selection

Controller Options

AXI Parameter

Memory Options

FPGA Options

Extended FPGA Options

IO Planning Options

Pin Selection

XILINX<sup>®</sup>

Pin Selection For Controller 0 - DDR3 SDRAM

	Signal Name	Bank Number	Byte Number	Pin Number	IO Standard
1	ddr3_dq[0]	35	T3	P6	SSTL15
2	ddr3_dq[1]	35	T3	R1	SSTL15
3	ddr3_dq[2]	35	T3	M5	SSTL15
4	ddr3_dq[3]	35	T3	N4	SSTL15
5	ddr3_dq[4]	35	T3	N5	SSTL15
6	ddr3_dq[5]	35	T3	N2	SSTL15
7	ddr3_dq[6]	35	T3	M6	SSTL15
8	ddr3_dq[7]	35	T3	P1	SSTL15
9	ddr3_dq[8]	35	T2	L3	SSTL15
10	ddr3_dq[9]	35	T2	J4	SSTL15
11	ddr3_dq[10]	35	T2	M3	SSTL15
12	ddr3_dq[11]	35	T2	K4	SSTL15
13	ddr3_dq[12]	35	T2	M2	SSTL15

INFO: Press **Validate** to proceed.

Validate

Read XDC/UCF

Save Pin Out

User Guide

< Back

Next >

Cancel

继续 Next

VIVADO<sup>®</sup>

HLx Editions

Pin Compatible FPGAs

Memory Selection

Controller Options

AXI Parameter

Memory Options

FPGA Options

Extended FPGA Options

IO Planning Options

Pin Selection

System Signals Selection

Summary

Simulation Options

XILINX<sup>®</sup>

System Signals Selection

Select the system pins below appropriately for the interface. Customization of these pins can also be made in the XDC after the design is generated. For more information see [UG586 Bank and Pin rules](#).

System Clock and Reference Clock pin selections will not be visible if the 'No Buffer' option was selected in the FPGA Options page.

System Signals

These signals may be connected internally to other logic or brought out to a pin.

- **sys\_rst**: This input signal is used to reset the interface.
- **init\_calib\_complete**: This signal indicates that the interface has completed calibration and memory initialization and is ready for commands. LOC constraint will be generated in XDC for Example design only based on "Pin Number" selection below.
- **error**: This output signal indicates that the traffic generator in the Example Design has detected a data mismatch. This signal does not exist in the User Design.

Signal Name	Bank Number	Pin Number
sys_rst	Select Bank	No connect
init_calib_complete	Select Bank	No connect
tg_compare_error	Select Bank	No connect

All pins must be constrained to specific locations in order to generate a bit file in the implementation phase (this is not required for simulation).

User Guide

< Back

Next >

Cancel

继续 Next

VIVADO  
HLx Editions

Pin Compatible FPGAs

Memory Selection

Controller Options

AXI Parameter

Memory Options

FPGA Options

Extended FPGA Options

IO Planning Options

Pin Selection

System Signals Selection

Summary

Simulation Options

PCB information

Design Notes

XILINX®

Vivado Project Options:

Target Device : xc7a100t-fgg484

Speed Grade : -2

HDL : verilog

Synthesis Tool : VIVADO

If any of the above options are incorrect, please click on "Cancel", change the CORE Generator Project Options, and restart

MIG Output Options:

Module Name : mig\_7series\_0

No of Controllers : 1

Selected Compatible Device(s) : --

FPGA Options:

System Clock Type : No Buffer

Reference Clock Type : Use System Clock

Debug Port : OFF

Internal Vref : disabled

IO Power Reduction : ON

XADC instantiation in MIG : Enabled

Extended FPGA Options:

DCI for DQ,DQS/DQS#,DM : enabled

Internal Termination (HR Banks) : 50 Ohms

Controller 0

Controller Options :

Memory : DDR3\_SDRAM

Interface : NATIVE

Design Clock Frequency : 2500 ps (400.00 MHz)

Phy to Controller Clock Ratio : 4:1

Input Clock Period : 4999 ps

CLKFBOUNT\_MULT (PLL) : 4

DIVCLK\_DIVIDE (PLL) : 1

VCC\_AUX\_IO : 1.8V

Memory Type : Components

Memory Part : MT41KL28M16XX-15E

Equivalent Part(s) : --

Data Width : 32

ECC : Disabled

Data Mask : enabled

ORDERING : Normal

Print

User Guide

< Back

Next >

Cancel

选择 Accept, 继续 Next

VIVADO  
HLx Editions

Pin Compatible FPGAs

Memory Selection

Controller Options

AXI Parameter

Memory Options

FPGA Options

Extended FPGA Options

IO Planning Options

Pin Selection

System Signals Selection

Summary

Simulation Options

PCB information

Design Notes

XILINX®

Micron Technology, Inc. Simulation Model License Agreement

PLEASE READ THIS SIMULATION MODEL LICENSE AGREEMENT ("AGREEMENT") FROM MICRON TECHNOLOGY, INC. ("MTI") CAREFULLY BEFORE INSTALLING OR USING THIS SIMULATION MODEL (THE "MODEL"). BY INSTALLING OR USING THE MODEL, YOU ARE ACCEPTING AND AGREEING TO THE TERMS AND CONDITIONS OF THIS AGREEMENT. IF YOU DO NOT AGREE WITH THE TERMS AND CONDITIONS OF THIS AGREEMENT, THEN DO NOT INSTALL OR USE THE MODEL.

SOFTWARE LICENSE: You acknowledge and agree that it is your sole responsibility to obtain the appropriate license or permission from the owner(s) of the software platform(s) that are necessary for you to operate the Model. MTI is under no obligation whatsoever to offer, provide or secure such license or permission for you.

MODEL LICENSE: MTI hereby grants to you the right to install, use and modify the Model solely for testing the Model and designing your product(s) in connection with the Model. You shall not use the Model or any modifications for any other purpose, and shall not copy, rent, or lease the Model or the modifications to any third party. MTI may make changes to the Model at any time without notice to you. MTI is under no obligation whatsoever to update, maintain, or provide new versions or other support for the Model.

OWNERSHIP OF MATERIALS: You acknowledge and agree that the Model is proprietary property of MTI and is protected by United States copyright law and international treaty provisions. The Model may not be copied, reproduced, published, uploaded, posted, transmitted, or distributed in any way without MTI's prior written permission. Except as expressly provided herein, MTI does not grant any express or implied right to you under any patents, copyrights, trademarks, or trade secret information. This Agreement does not convey to you an interest in or to the Model, but only a limited right to use and modify the Model in accordance with the terms of this Agreement.

DISCLAIMER OF WARRANTY: THE MODEL IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND. MTI EXPRESSLY DISCLAIMS ALL WARRANTIES EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, NON-INFRINGEMENT OF THIRD PARTY RIGHTS, AND ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE. MTI DOES NOT WARRANT THAT THE MODEL WILL MEET YOUR REQUIREMENTS, OR THAT THE OPERATION OF THE MODEL WILL BE UNINTERRUPTED OR ERROR-FREE. FURTHERMORE, MTI DOES NOT MAKE ANY REPRESENTATIONS REGARDING THE USE OR THE RESULTS OF THE USE OF THE MODEL IN TERMS OF ITS CORRECTNESS, ACCURACY, RELIABILITY, OR OTHERWISE. THE ENTIRE RISK ARISING OUT OF USE OR PERFORMANCE OF THE MODEL REMAINS WITH YOU. IN NO EVENT SHALL MTI, ITS AFFILIATED COMPANIES OR THEIR SUPPLIERS BE LIABLE FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL, INCIDENTAL, OR SPECIAL DAMAGES (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF PROFITS, BUSINESS INTERRUPTION, OR LOSS OF INFORMATION) ARISING OUT OF YOUR USE OF OR INABILITY TO USE THE MODEL, EVEN IF MTI HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Because some jurisdictions prohibit the exclusion or limitation of liability for consequential or incidental damages, the above limitation may not apply to you.

Check Accept or Decline to proceed. By clicking Accept, memory model will be output in the simulation directory. By clicking Decline, a memory model must be acquired and configured appropriately.

Print

☒ Accept ☐ Decline

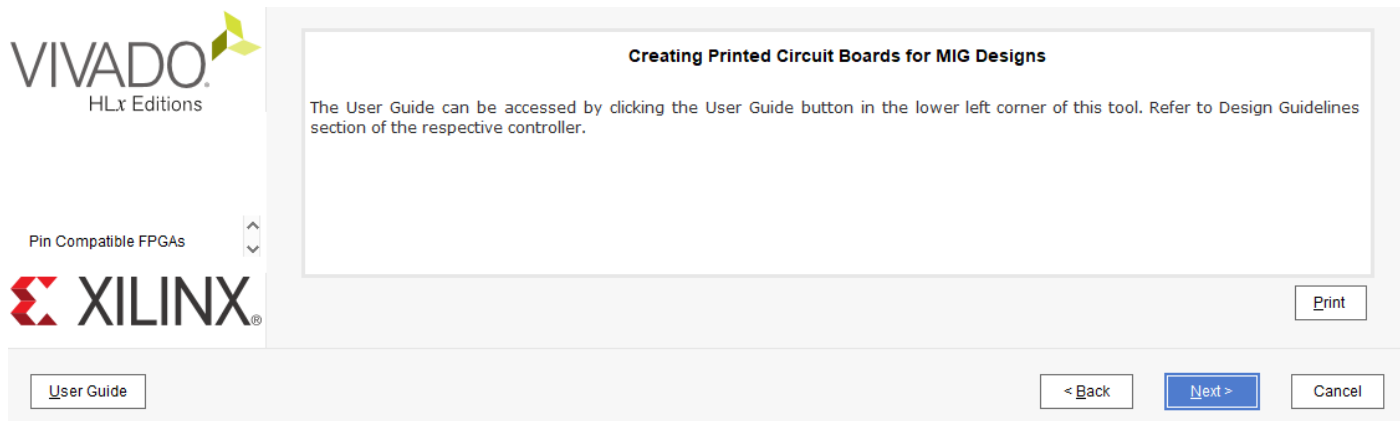
User Guide

< Back

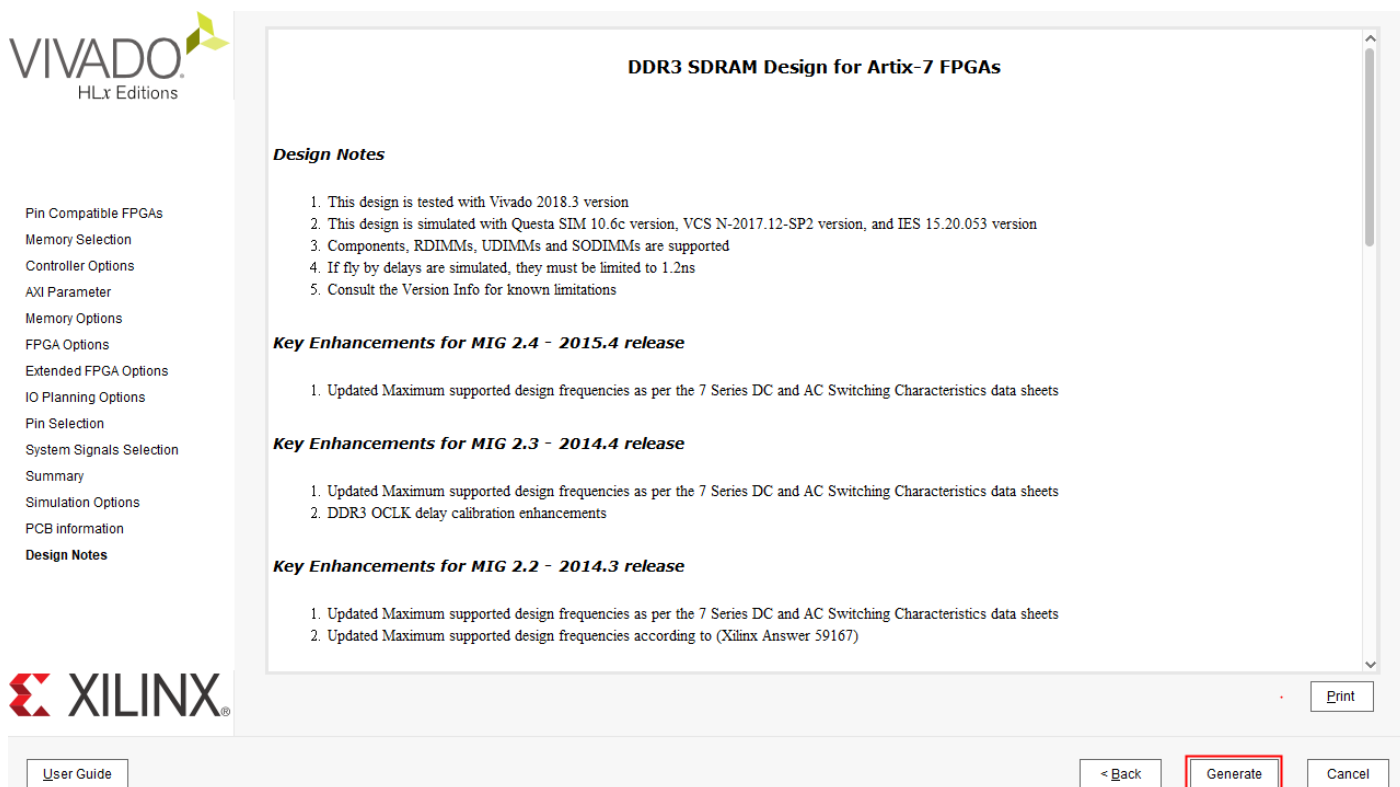
Next >

Cancel

继续 Next

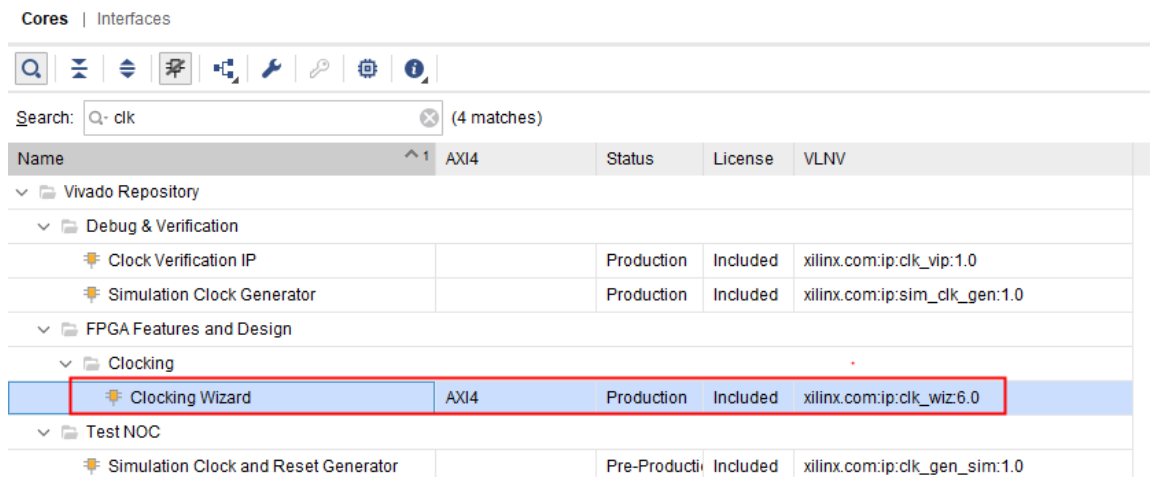


Generate



## 1.5.2 添加时钟 PLL

针对 A7 开发板输入单端 50M 时钟需要经过 PLL 倍频到 200M 给 MIG 时钟



Component Name

Clocking Options

Output Clocks

Port Renaming

MMCM Settings

Summary

**Clock Monitor**

☐ Enable Clock Monitoring

**Primitive**

☒ MMCM ☐ PLL

**Clocking Features**

☒ Frequency Synthesis ☐ Minimize Power ☐ Phase Alignment ☐ Spread Spectrum ☐ Dynamic Reconfig ☐ Dynamic Phase Shift ☐ Safe Clock Startup

**Jitter Optimization**

☐ Balanced ☒ Minimize Output Jitter ☐ Maximize Input Jitter filtering

**Dynamic Reconfig Interface Options**

☒ AXI4Lite ☐ DRP ☐ Phase Duty Cycle Config ☐ Write DRP registers

**Input Clock Information**

	Input Clock	Port Name	Input Frequency(MHz)		Jitter Options	Input Jitter	Source
<input checked="" type="checkbox"/>	Primary	clk_in1	50.000	10.000 - 800.000	UI	0.010	Global buffer
<input type="checkbox"/>	Secondary	clk_in2	100.000	21.053 - 50.526		0.010	Single ended clock capable pin

Component Name: clk\_wiz\_0

Clocking Options	Output Clocks	Port Renaming	MMCM Settings	Summary
<input checked="" type="checkbox"/> clk_out1	clk_out1	200.000	200.000000	0.000
<input type="checkbox"/> clk_out2	clk_out2	400.000	N/A	0.000
<input type="checkbox"/> clk_out3	clk_out3	100.000	N/A	0.000
<input type="checkbox"/> clk_out4	clk_out4	100.000	N/A	0.000
<input type="checkbox"/> clk_out5	clk_out5	100.000	N/A	0.000
<input type="checkbox"/> clk_out6	clk_out6	100.000	N/A	0.000
<input type="checkbox"/> clk_out7	clk_out7	100.000	N/A	0.000

☐ USE CLOCK SEQUENCING

**Clocking Feedback**

Output Clock	Sequence Number
clk_out1	1
clk_out2	1
clk_out3	1
clk_out4	1
clk_out5	1
clk_out6	1
clk_out7	1

**Source**

☒ Automatic Control On-Chip  
☐ Automatic Control Off-Chip  
☐ User-Controlled On-Chip  
☐ User-Controlled Off-Chip

**Signaling**

☒ Single-ended  
☐ Differential

**Enable Optional Inputs / Outputs for MMCM/PLL**

☒ reset ☐ power\_down ☐ input\_clk\_stopped  
☒ locked ☐ clkfbstopped

**Reset Type**

☒ Active High ☐ Active Low

### 1.5.3 编写测试代码

以下以 MA703FA-100T 为测试样板说明。编写 mig\_ddr\_test.v 测试文件通过访问 MIG 用户接口对 DDR 进行测试，注意：以下代码中已经添加(\*mark\_debug = "true"\*)在线仿真必要的信号预定义。对于 MA703FA-35T 以下代码位宽有差异，具体打开配套工程代码阅读。

```

`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/*
Company : Liyang Milian Electronic Technology Co., Ltd.
Brand: 米联客 milianke(msxbo)
Technical forum: uisrc.com
taobao: osrc.taobao.com
Create Date: 2020/07/02
Module Name: mig_ddr_test
Description:
Copyright: Copyright (c) msxbo
Revision: 1.0

```

Signal description:

- 1) \_i input
- 2) \_o output
- 3) \_n activ low
- 4) \_dg debug signal
- 5) \_r delay or register
- 6) \_s state mechine

\*/

////////////////////////////////////

module mig\_ddr\_test

```
(
    inout [31:0]          ddr3_dq,
    inout [3:0]           ddr3_dqs_n,
    inout [3:0]           ddr3_dqs_p,
    output [13:0]          ddr3_addr,
    output [2:0]           ddr3_ba,
    output                ddr3_ras_n,
    output                ddr3_cas_n,
    output                ddr3_we_n,
    output                ddr3_reset_n,
    output [0:0]           ddr3_ck_p,
    output [0:0]           ddr3_ck_n,
    output [0:0]           ddr3_cke,
    output [0:0]           ddr3_cs_n,
    output [3:0]           ddr3_dm,
    output [0:0]           ddr3_odt,
    input                 sysclk_i,
    output                tg_compare_error,
    // output              breath_light,
    output                init_calib_complete
);
```

wire sys\_rst = 1'b0;

wire clk\_200m, locked;

clk\_wiz\_0 clk\_wiz\_inst(.clk\_out1(clk\_200m),.reset(sys\_rst),.locked(locked),.clk\_in1(sysclk\_i));

```
localparam ADDR_WIDTH      = 28;
localparam DATA_WIDTH     = 32;
localparam PAYLOAD_WIDTH   = DATA_WIDTH;
localparam BURST_LENGTH    = 8;
localparam APP_DATA_WIDTH  = 256;
localparam APP_MASK_WIDTH  = APP_DATA_WIDTH / 8;
```

```
wire [ADDR_WIDTH-1:0]      app_addr;
wire [2:0]                 app_cmd;
wire                      app_en;
wire                      app_rdy;
wire [APP_DATA_WIDTH-1:0] app_rd_data;
```

```

wire                app_rd_data_end;
wire                app_rd_data_valid;
wire [APP_DATA_WIDTH-1:0] app_wdf_data;
wire                app_wdf_end;
wire                app_wdf_rdy;
wire                app_sr_active;
wire                app_ref_ack;
wire                app_zq_ack;
wire                app_wdf_wren;
wire [11:0]         device_temp;
wire                ui_clk;
wire                ui_rst;

localparam [1:0] IDLE      =2'd0;
localparam [1:0] WRITE    =2'd1;
localparam [1:0] WAIT     =2'd2;
localparam [1:0] READ     =2'd3;
localparam [2:0] CMD_WRITE =3'd0;
localparam [2:0] CMD_READ  =3'd1;
//localparam TEST_DATA_RANGE =24'd16777210;//部分测试
localparam TEST_DATA_RANGE =24'd1000;//部分测试

(*mark_debug = "true"*) reg [1:0] state=0;
reg [23:0] Count_64=0;// 128M*2*16/256
reg [ADDR_WIDTH-1:0] app_addr_begin=0;
(*mark_debug = "true"*) wire tg_compare_error;

assign app_wdf_end      =app_wdf_wren;//两个相等即可
assign app_en           =(state==WRITE) ? (app_rdy&&app_wdf_rdy) : ((state==READ)&&app_rdy);
assign app_wdf_wren      =(state==WRITE) ? (app_rdy&&app_wdf_rdy) : 1'b0;
assign app_cmd           =(state==WRITE) ? CMD_WRITE : CMD_READ;
assign app_addr          =app_addr_begin;
assign app_wdf_data      ={
Count_64[7:0],Count_64[7:0],Count_64[7:0],Count_64[7:0],Count_64[7:0],Count_64[7:0],Count_64[7:0],Count_64[7:0],
Count_64[7:0],Count_64[7:0],Count_64[7:0],Count_64[7:0],Count_64[7:0],Count_64[7:0],Count_64[7:0],Count_64[7:0],
Count_64[7:0],Count_64[7:0],Count_64[7:0],Count_64[7:0],Count_64[7:0],Count_64[7:0],Count_64[7:0],Count_64[7:0],
Count_64[7:0],Count_64[7:0],Count_64[7:0],Count_64[7:0],Count_64[7:0],Count_64[7:0],Count_64[7:0],Count_64[7:0]
};//写入的数据是计数器

always@(posedge ui_clk)
    if(ui_rst&!init_calib_complete)//
        begin
            state <=IDLE;

```

```

        app_addr_begin          <=28' d0;
        Count_64                <=24' d0;
    end
else case(state)
    IDLE:    begin
        state          <=WRITE;
        if(app_addr_begin >= 24' d16777210)
            app_addr_begin          <=28' d0;
            Count_64                <=24' d0;
        end
        WRITE:    begin//写整片的 DDR3
            state          <=(Count_64==TEST_DATA_RANGE)&&app_rdy&&app_wdf_rdy ? WAIT:state;//最后一个地址写完之后跳出状态
            Count_64          <=app_rdy&&app_wdf_rdy?(Count_64+24' d1):Count_64;
            app_addr_begin    <=app_rdy&&app_wdf_rdy?(app_addr_begin+28' d8):app_addr_begin;//跳到下一个(8*32=256) bit 数据地址
        end
        WAIT:    begin
            state          <=READ;
            Count_64          <=24' d0;
            app_addr_begin    <=28' d0;
        end
        READ:    begin//读整片的 DDR3
            state          <=(Count_64==TEST_DATA_RANGE)&&app_rdy? IDLE:state;
            Count_64          <=app_rdy?(Count_64+24' d1):Count_64;
            app_addr_begin    <=app_rdy?(app_addr_begin+28' d8):app_addr_begin;
        end
    default:begin
        state          <=IDLE;
        app_addr_begin    <=28' d0;
        Count_64          <=24' d0;
    end
endcase

(*mark_debug = "true"*) (* KEEP = "TRUE" *) reg [63:0]app_rd_data_r =0;
(*mark_debug = "true"*) (* KEEP = "TRUE" *) reg app_rd_data_valid_r =0;

always @(posedge ui_clk) begin
    app_rd_data_r <= app_rd_data[63:0];
    app_rd_data_valid_r <= app_rd_data_valid;
end

//16bit count used for comparation
reg [7:0] count_temp =0;
always @(posedge ui_clk) begin
    if(app_rd_data_valid_r)
        count_temp<= count_temp + 1'b1;
    else if(state==WAIT)count_temp <= 8' d0;
end

```



```

end

//compare data read from mig
(*mark_debug = "true"*) wire [63:0]cm_data;
assign cm_data = {count_temp,count_temp,count_temp,count_temp,count_temp,count_temp,count_temp,count_temp};
assign tg_compare_error=(app_rd_data_valid_r&&(cm_data!=app_rd_data_r));

mig_7series_0 u_mig_7series_0 (
    // Memory interface ports
    .ddr3_addr          (ddr3_addr), // output [13:0]          ddr3_addr
    .ddr3_ba            (ddr3_ba), // output [2:0]          ddr3_ba
    .ddr3_cas_n         (ddr3_cas_n), // output                ddr3_cas_n
    .ddr3_ck_n          (ddr3_ck_n), // output [0:0]          ddr3_ck_n
    .ddr3_ck_p          (ddr3_ck_p), // output [0:0]          ddr3_ck_p
    .ddr3_cke           (ddr3_cke), // output [0:0]          ddr3_cke
    .ddr3_ras_n         (ddr3_ras_n), // output                ddr3_ras_n
    .ddr3_reset_n       (ddr3_reset_n), // output                ddr3_reset_n
    .ddr3_we_n          (ddr3_we_n), // output                ddr3_we_n
    .ddr3_dq            (ddr3_dq), // inout [31:0]          ddr3_dq
    .ddr3_dqs_n         (ddr3_dqs_n), // inout [3:0]          ddr3_dqs_n
    .ddr3_dqs_p         (ddr3_dqs_p), // inout [3:0]          ddr3_dqs_p
    .ddr3_cs_n          (ddr3_cs_n), // output [0:0]          ddr3_cs_n
    .ddr3_dm            (ddr3_dm), // output [3:0]          ddr3_dm
    .ddr3_odt           (ddr3_odt), // output [0:0]          ddr3_odt
    .init_calib_complete (init_calib_complete), // output                init_calib_complete
    // Application interface ports
    .app_addr           (app_addr), // input [27:0]          app_addr
    .app_cmd            (app_cmd), // input [2:0]           app_cmd
    .app_en             (app_en), // input                 app_en
    .app_wdf_data       (app_wdf_data), // input [255:0]        app_wdf_data
    .app_wdf_end        (app_wdf_end), // input                 app_wdf_end
    .app_wdf_wren       (app_wdf_wren), // input                 app_wdf_wren
    .app_rd_data        (app_rd_data), // output [255:0]        app_rd_data
    .app_rd_data_end    (app_rd_data_end), // output                app_rd_data_end
    .app_rd_data_valid  (app_rd_data_valid), // output                app_rd_data_valid
    .app_rdy            (app_rdy), // output                app_rdy
    .app_wdf_rdy        (app_wdf_rdy), // output                app_wdf_rdy
    .app_sr_req         (1'b0),
    .app_ref_req        (1'b0),
    .app_zq_req         (1'b0),
    .app_sr_active      (app_sr_active), // output                app_sr_active
    .app_ref_ack        (app_ref_ack), // output                app_ref_ack
    .app_zq_ack         (app_zq_ack), // output                app_zq_ack
    .ui_clk             (ui_clk), // output                ui_clk
    .ui_clk_sync_rst    (ui_rst), // output                ui_clk_sync_rst
    .app_wdf_mask       (32'd0), // input [31:0]          app_wdf_mask
    // System Clock Ports
    .sys_clk_i          (clk_200m),

```

```
.sys_rst (locked), // input sys_rst
.device_temp (device_temp)
);

endmodule
```

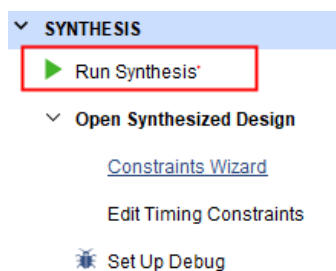
## 1.5.4 搭建完成



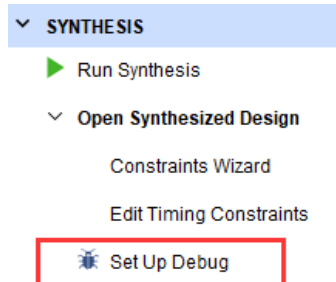
## 1.6 在线仿真

### 1.6.1 添加仿真信号

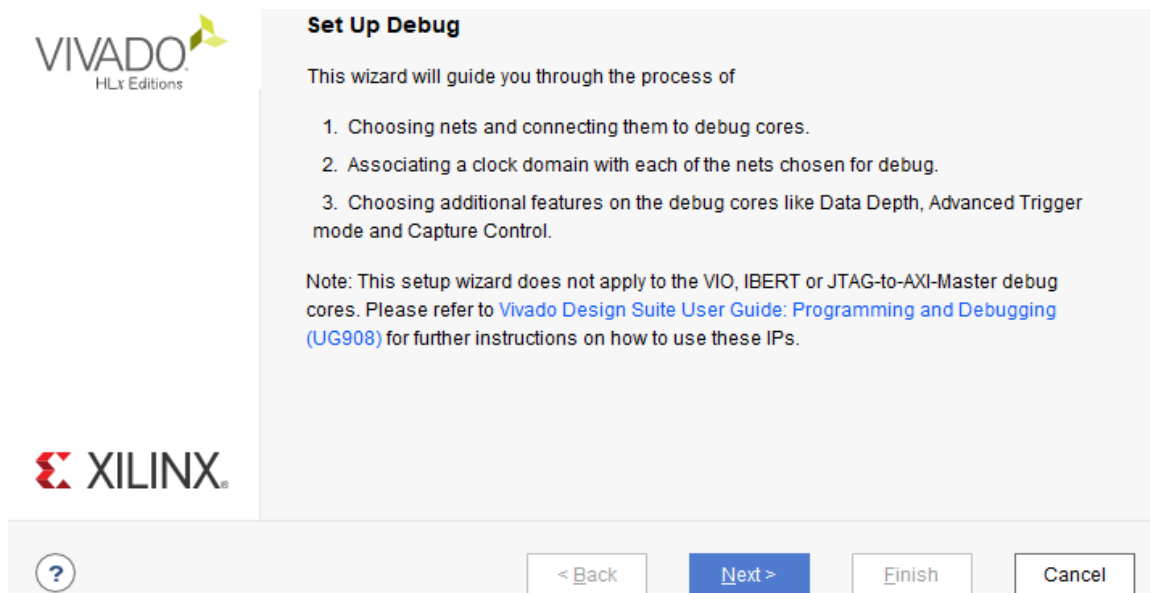
编译并下载程序，对于添加(\*mark\_debug = "true"\*)关键词的在线仿真信号，需要正确方法才能添加仿真信号。具体也可以阅读“米联客(MSXBO)2020 版 FPGA 课程”（第 30 课）。先综合



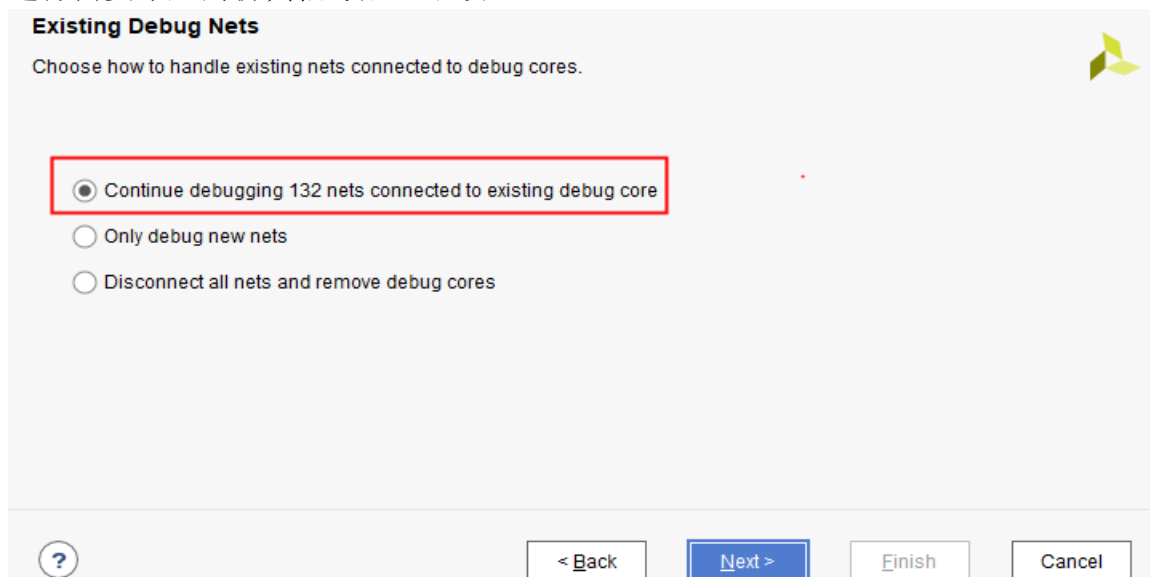
综合完成后运行 Set Up Debug 添加标记的仿真信号



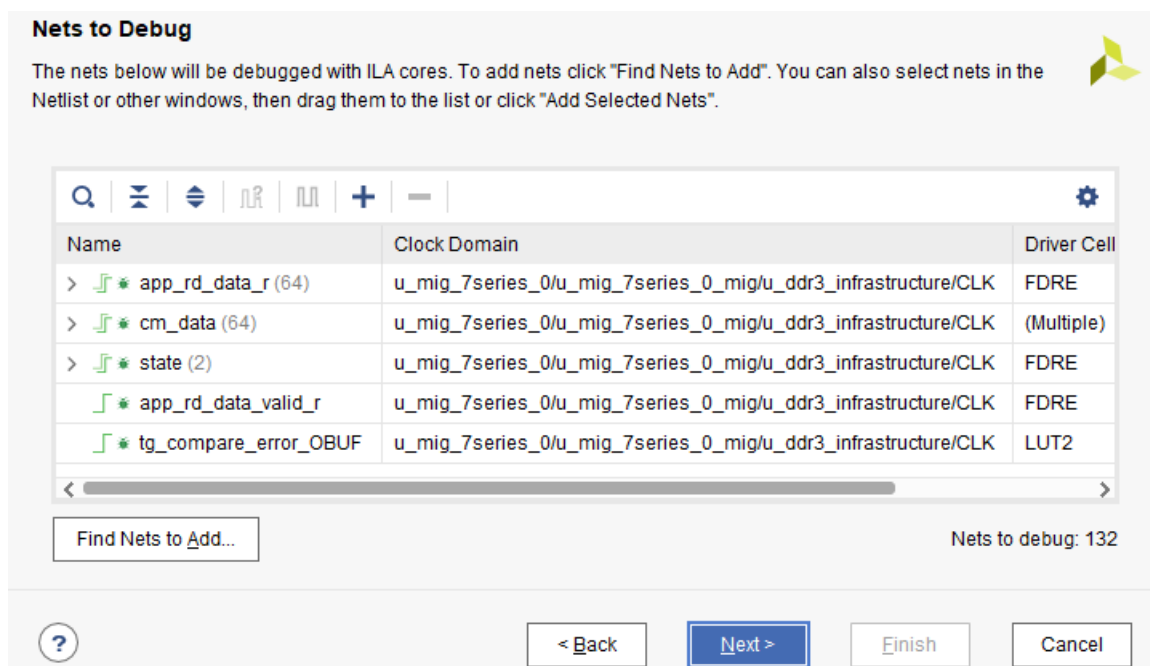
继续 Next



选择需要添加的仿真信号后，继续 Next



查看添加的仿真信号，继续 Next



设置采样深度，深度越大观察的时间轴越长，但是需要消耗 FPGA 内部的 BRAM

**ILA Core Options**

Choose features for the ILA debug cores.

Sample of data depth: 4096

Input pipe stages: 0

**Trigger and Storage Settings**

☐ Capture control

☐ Advanced trigger

? < Back Next > Finish Cancel

完成

**VIVADO**  
HLx Editions

**XILINX**

**Set up Debug Summary**

- 1 debug core will be removed: u\_ila\_0
- 1 debug core will be created
- Found 1 clock

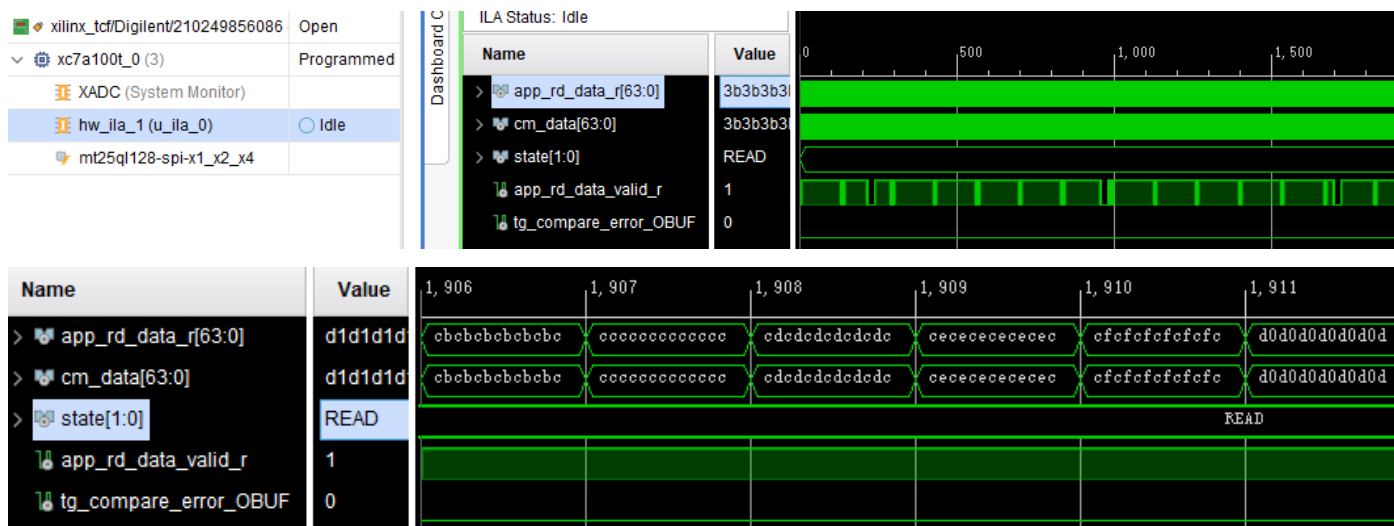
☒ Open in Debug layout

To apply the above changes, click Finish

? < Back Next > Finish Cancel

## 1.6.2 在线仿真结果

编译并且产生 bit 文件下载到 FPGA 测试



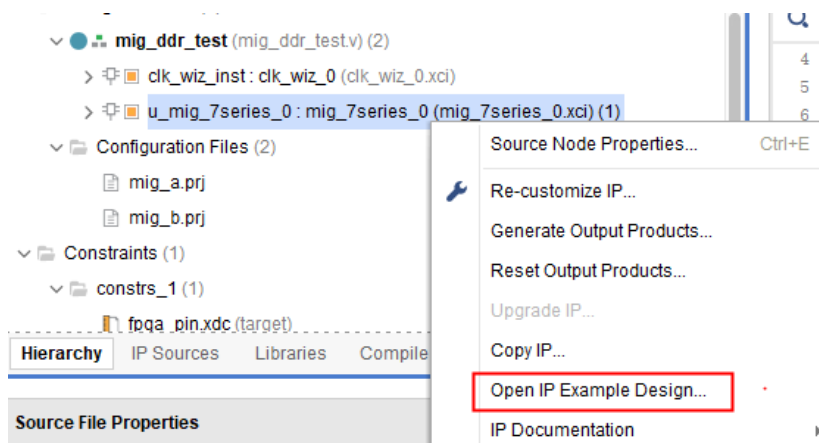
## 1.7 软件仿真

通常情况下，对于 FPGA 编程，采用软件仿真的方式要比硬件仿真具有更高的效率，更好的发现问题。虽然我们上面直接在 FPGA 上验证没有问题，但是仍然有必要掌握下软件仿真 MIG 实现读写 DDR

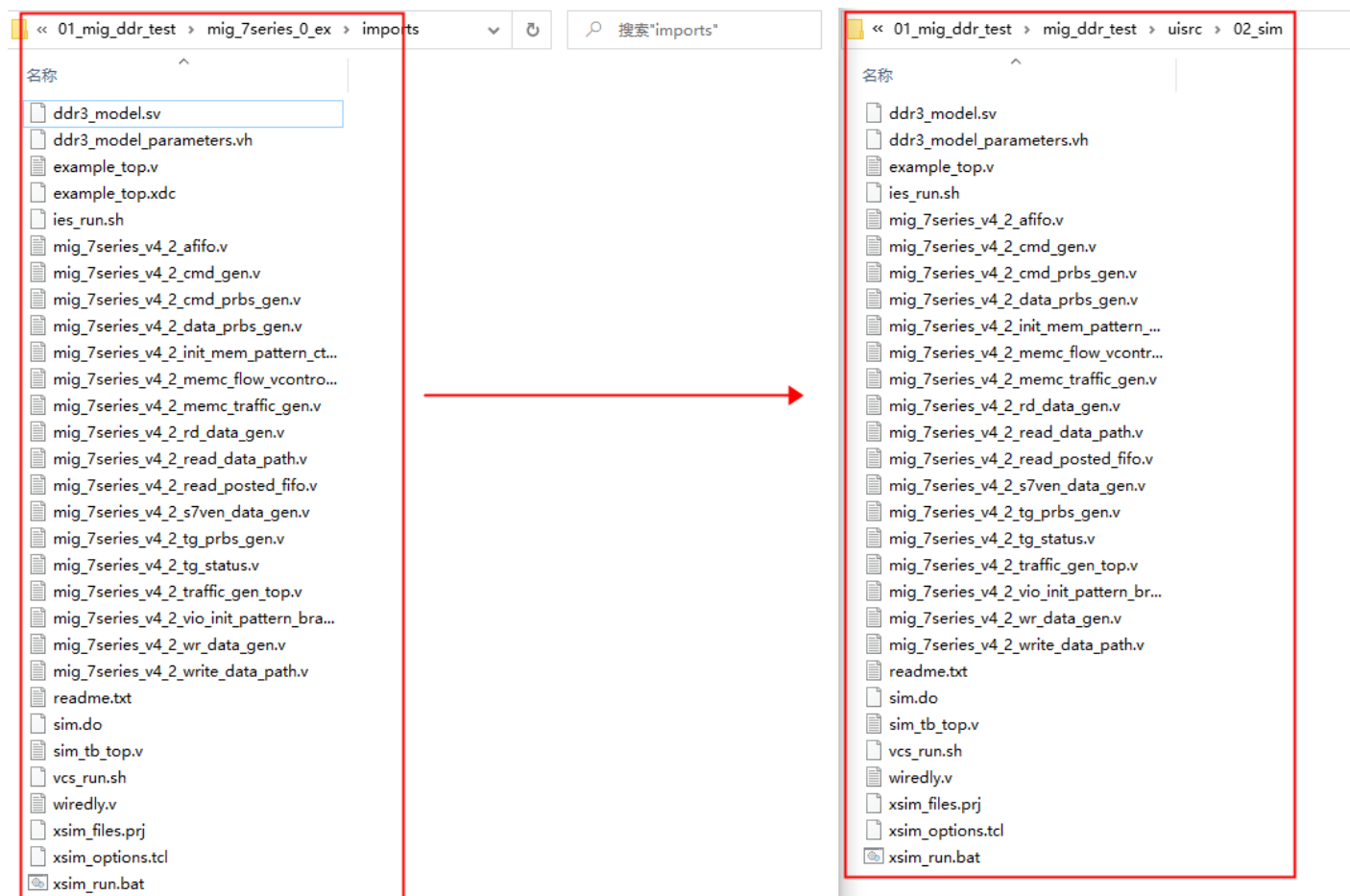
### 1.7.1 用自带 demo 产生测试工程

VIVADO 有一个很好用的功能就是可以针对绝大部分 IP 自动产生测试工程。当然软件自动产生的测试工程代码阅读理解起来惨不忍睹，我们主要是未来参数必要的仿真测试文件。

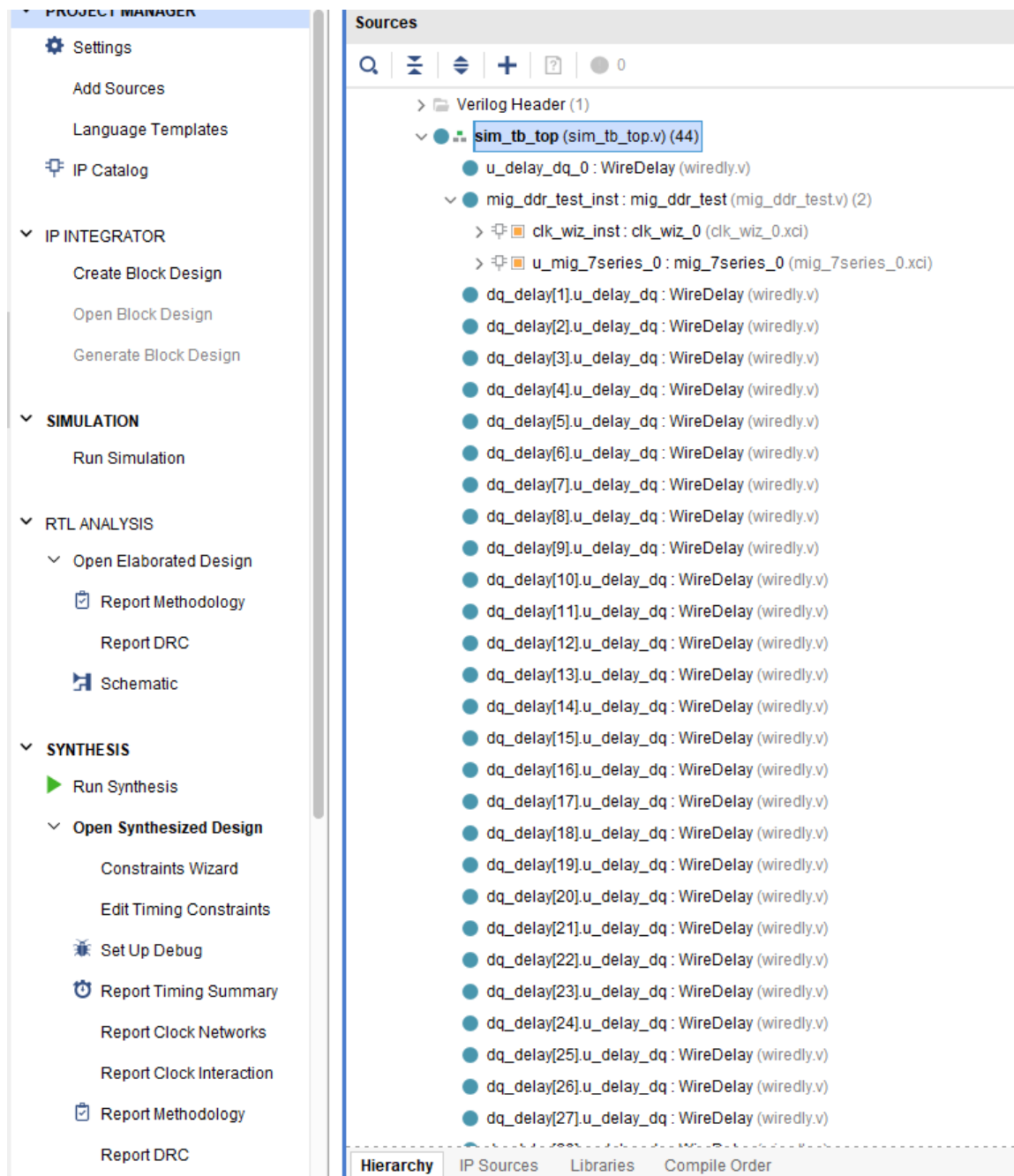
右击 MIG IP 选择 Open IP Example Design



复制自动产生测试代码下仿真文件到 uisrc/sim 路径



## 1.7.2 添加仿真文件并测试



## 1.7.3 修改必要的仿真文件参数

1)、修改输入时钟频率，对于 MA703 开发板是 50M 输入时钟：

***parameter CLKIN\_PERIOD = 20000;***

2)、修改调用 mig\_ddr\_test.v 模块接口：

***mig\_ddr\_test mig\_ddr\_test\_inst***

```
(
    .ddr3_dq          (ddr3_dq_fpga),
    .ddr3_dqs_n       (ddr3_dqs_n_fpga),
    .ddr3_dqs_p       (ddr3_dqs_p_fpga),
    .ddr3_addr        (ddr3_addr_fpga),
```

```

.ddr3_ba      (ddr3_ba_fpga),
.ddr3_ras_n   (ddr3_ras_n_fpga),
.ddr3_cas_n   (ddr3_cas_n_fpga),
.ddr3_we_n    (ddr3_we_n_fpga),
.ddr3_reset_n (ddr3_reset_n),
.ddr3_ck_p    (ddr3_ck_p_fpga),
.ddr3_ck_n    (ddr3_ck_n_fpga),
.ddr3_cke     (ddr3_cke_fpga),
.ddr3_cs_n    (ddr3_cs_n_fpga),
.ddr3_dm      (ddr3_dm_fpga),
.ddr3_odt     (ddr3_odt_fpga),
.sysclk_i     (sys_clk_i),
.init_calib_complete (init_calib_complete),
.tg_compare_error (tg_compare_error)
);

```

3)、由于仿真速度很慢，整个 DDR 空间仿真会消耗大量时间所有修改 mig\_ddr\_test.v 中测试的地址范围：

**localparam TEST\_DATA\_RANGE =24'd1000; // 部分测试**

最终完整修改好的 sim\_tb\_top.v 仿真文件如下

```

////////////////////////////////////////////////////////////////
/*
Company : Liyang Milian Electronic Technology Co., Ltd.
Brand: 米联客 milianke(msxbo)
Technical forum: uisrc.com
taobao: osrc.taobao.com
Create Date: 2020/07/02
Module Name: sim_tb_top
Description:
Copyright: Copyright (c) msxbo
Revision: 1.0
Signal description:
1) _i input
2) _o output
3) _n activ low
4) _dg debug signal
5) _r delay or register
6) _s state mechine
*/
////////////////////////////////////////////////////////////////

`timescale 1ps/100fs

module sim_tb_top;

    //*****
    // Traffic Gen related parameters
    //*****

    parameter SIMULATION          = "TRUE";
    parameter PORT_MODE           = "BI_MODE";

```

```

parameter DATA_MODE          = 4'b0010;
parameter TST_MEM_INSTR_MODE  = "R_W_INSTR_MODE";
parameter EYE_TEST            = "FALSE";
                                // set EYE_TEST = "TRUE" to probe memory
                                // signals. Traffic Generator will only
                                // write to one single location and no
                                // read transactions will be generated.

parameter DATA_PATTERN       = "DGEN_ALL";
                                // For small devices, choose one only.
                                // For large device, choose "DGEN_ALL"
                                // "DGEN_HAMMER", "DGEN_WALKING1",
                                // "DGEN_WALKING0", "DGEN_ADDR", "
                                // "DGEN_NEIGHBOR", "DGEN_PRBS", "DGEN_ALL"

parameter CMD_PATTERN         = "CGEN_ALL";
                                // "CGEN_PRBS", "CGEN_FIXED", "CGEN_BRAM",
                                // "CGEN_SEQUENTIAL", "CGEN_ALL"

parameter BEGIN_ADDRESS       = 32'h00000000;
parameter END_ADDRESS         = 32'h00000fff;
parameter PRBS_EADDR_MASK_POS = 32'hff000000;

//*****
// The following parameters refer to width of various ports
//*****

parameter COL_WIDTH           = 10;
                                // # of memory Column Address bits.

parameter CS_WIDTH            = 1;
                                // # of unique CS outputs to memory.

parameter DM_WIDTH            = 4;
                                // # of DM (data mask)

parameter DQ_WIDTH            = 32;
                                // # of DQ (data)

parameter DQS_WIDTH           = 4;
parameter DQS_CNT_WIDTH       = 2;
                                // = ceil(log2(DQS_WIDTH))

parameter DRAM_WIDTH          = 8;
                                // # of DQ per DQS

parameter ECC                  = "OFF";
parameter RANKS                = 1;
                                // # of Ranks.

parameter ODT_WIDTH           = 1;
                                // # of ODT outputs to memory.

parameter ROW_WIDTH           = 14;
                                // # of memory Row Address bits.

parameter ADDR_WIDTH          = 28;
                                // # = RANK_WIDTH + BANK_WIDTH
                                //      + ROW_WIDTH + COL_WIDTH;
                                // Chip Select is always tied to low for
                                // single rank devices

```



```

//*****

// The following parameters are mode register settings

//*****

parameter BURST_MODE          = "8";

                                // DDR3 SDRAM:
                                // Burst Length (Mode Register 0).
                                // # = "8", "4", "OTF".
                                // DDR2 SDRAM:
                                // Burst Length (Mode Register).
                                // # = "8", "4".

parameter CA_MIRROR           = "OFF";

                                // C/A mirror opt for DDR3 dual rank

//*****

// The following parameters are multiplier and divisor factors for PLLE2.
// Based on the selected design frequency these parameters vary.
//*****

parameter CLKIN_PERIOD        = 20000;

                                // Input Clock Period

//*****

// Simulation parameters
//*****

parameter SIM_BYPASS_INIT_CAL = "FAST";

                                // # = "SIM_INIT_CAL_FULL" - Complete
                                // memory init &
                                // calibration sequence
                                // # = "SKIP" - Not supported
                                // # = "FAST" - Complete memory init & use
                                // abbreviated calib sequence

//*****

// IODELAY and PHY related parameters
//*****

parameter TCQ                  = 100;

//*****

// IODELAY and PHY related parameters
//*****

parameter RST_ACT_LOW          = 1;

                                // =1 for active low reset,
                                // =0 for active high.

//*****

// Referece clock frequency parameters
//*****

parameter REFCLK_FREQ          = 200.0;

                                // IODELAYCTRL reference clock frequency

```

```

//*****

// System clock frequency parameters
//*****

parameter tCK                = 2500;

                                // memory tCK paramter.

                                // # = Clock Period in pS.
parameter nCK_PER_CLK        = 4;

                                // # of memory CKs per fabric CLK


//*****

// Debug and Internal parameters
//*****

parameter DEBUG_PORT          = "OFF";

                                // # = "ON" Enable debug signals/controls.
                                //   = "OFF" Disable debug signals/controls.

//*****

// Debug and Internal parameters
//*****

parameter DRAM_TYPE           = "DDR3";


//*****//

// Local parameters Declarations
//*****//

localparam real TPROP_DQS      = 0.00;

                                // Delay for DQS signal during Write Operation
localparam real TPROP_DQS_RD   = 0.00;

                                // Delay for DQS signal during Read Operation
localparam real TPROP_PCB_CTRL = 0.00;

                                // Delay for Address and Ctrl signals
localparam real TPROP_PCB_DATA = 0.00;

                                // Delay for data signal during Write operation
localparam real TPROP_PCB_DATA_RD = 0.00;

                                // Delay for data signal during Read operation


localparam MEMORY_WIDTH        = 16;
localparam NUM_COMP             = DQ_WIDTH/MEMORY_WIDTH;
localparam ECC_TEST             = "OFF" ;
localparam ERR_INSERT = (ECC_TEST == "ON") ? "OFF" : ECC ;


localparam real REFCLK_PERIOD = (1000000.0/(2*REFCLK_FREQ));
localparam RESET_PERIOD = 200000; //in pSec
localparam real SYSCLK_PERIOD = tCK;

```

```

//*****//
// Wire Declarations
//*****//

reg                                sys_rst_n;
wire                               sys_rst;

reg                                sys_clk_i;

reg clk_ref_i;

wire                               ddr3_reset_n;
wire [DQ_WIDTH-1:0]               ddr3_dq_fpga;
wire [DQS_WIDTH-1:0]              ddr3_dqs_p_fpga;
wire [DQS_WIDTH-1:0]              ddr3_dqs_n_fpga;
wire [ROW_WIDTH-1:0]              ddr3_addr_fpga;
wire [3-1:0]                       ddr3_ba_fpga;
wire                               ddr3_ras_n_fpga;
wire                               ddr3_cas_n_fpga;
wire                               ddr3_we_n_fpga;
wire [1-1:0]                       ddr3_cke_fpga;
wire [1-1:0]                       ddr3_ck_p_fpga;
wire [1-1:0]                       ddr3_ck_n_fpga;

wire                               init_calib_complete;
wire                               tg_compare_error;
wire [(CS_WIDTH*1)-1:0] ddr3_cs_n_fpga;

wire [DM_WIDTH-1:0]               ddr3_dm_fpga;

wire [ODT_WIDTH-1:0]              ddr3_odt_fpga;

reg [(CS_WIDTH*1)-1:0] ddr3_cs_n_sdr_tm;

reg [DM_WIDTH-1:0]               ddr3_dm_sdr_tm;

reg [ODT_WIDTH-1:0]              ddr3_odt_sdr_tm;

wire [DQ_WIDTH-1:0]               ddr3_dq_sdr;
reg [ROW_WIDTH-1:0]               ddr3_addr_sdr [0:1];

```

```

reg [3-1:0]                ddr3_ba_sdram [0:1];
reg                    ddr3_ras_n_sdram;
reg                    ddr3_cas_n_sdram;
reg                    ddr3_we_n_sdram;
wire [(CS_WIDTH*1)-1:0] ddr3_cs_n_sdram;
wire [ODT_WIDTH-1:0]    ddr3_odt_sdram;
reg [1-1:0]             ddr3_cke_sdram;
wire [DM_WIDTH-1:0]     ddr3_dm_sdram;
wire [DQS_WIDTH-1:0]    ddr3_dqs_p_sdram;
wire [DQS_WIDTH-1:0]    ddr3_dqs_n_sdram;
reg [1-1:0]             ddr3_ck_p_sdram;
reg [1-1:0]             ddr3_ck_n_sdram;

//*****//

//*****//

// Reset Generation
//*****//

initial begin
    sys_rst_n = 1'b0;
    #RESET_PERIOD
    sys_rst_n = 1'b1;
end

assign sys_rst = RST_ACT_LOW ? sys_rst_n : ~sys_rst_n;

//*****//

// Clock Generation
//*****//

initial
    sys_clk_i = 1'b0;
always
    sys_clk_i = #(CLKIN_PERIOD/2.0) ~sys_clk_i;

initial
    clk_ref_i = 1'b0;
always
    clk_ref_i = #REFCLK_PERIOD ~clk_ref_i;

always @( * ) begin
    ddr3_ck_p_sdram    <=  #(TPROP_PCB_CTRL) ddr3_ck_p_fpga;
    ddr3_ck_n_sdram    <=  #(TPROP_PCB_CTRL) ddr3_ck_n_fpga;
    ddr3_addr_sdram[0] <=  #(TPROP_PCB_CTRL) ddr3_addr_fpga;
    ddr3_addr_sdram[1] <=  #(TPROP_PCB_CTRL) (CA_MIRROR == "ON") ?
                        { ddr3_addr_fpga[ROW_WIDTH-1:9],

```

```

                                ddr3_addr_fpga[7], ddr3_addr_fpga[8],
                                ddr3_addr_fpga[5], ddr3_addr_fpga[6],
                                ddr3_addr_fpga[3], ddr3_addr_fpga[4],
                                ddr3_addr_fpga[2:0] } :
                                ddr3_addr_fpga;

ddr3_ba_sdram[0]    <=  #(TPROP_PCB_CTRL) ddr3_ba_fpga;
ddr3_ba_sdram[1]    <=  #(TPROP_PCB_CTRL) (CA_MIRROR == "ON") ?
                                { ddr3_ba_fpga[3-1:2],
                                ddr3_ba_fpga[0],
                                ddr3_ba_fpga[1] } :
                                ddr3_ba_fpga;

ddr3_ras_n_sdram    <=  #(TPROP_PCB_CTRL) ddr3_ras_n_fpga;
ddr3_cas_n_sdram    <=  #(TPROP_PCB_CTRL) ddr3_cas_n_fpga;
ddr3_we_n_sdram     <=  #(TPROP_PCB_CTRL) ddr3_we_n_fpga;
ddr3_cke_sdram      <=  #(TPROP_PCB_CTRL) ddr3_cke_fpga;
end

always @( * )
    ddr3_cs_n_sdram_tmp    <=  #(TPROP_PCB_CTRL) ddr3_cs_n_fpga;
assign ddr3_cs_n_sdram =  ddr3_cs_n_sdram_tmp;

always @( * )
    ddr3_dm_sdram_tmp <=  #(TPROP_PCB_DATA) ddr3_dm_fpga;//DM signal generation
assign ddr3_dm_sdram = ddr3_dm_sdram_tmp;

always @( * )
    ddr3_odt_sdram_tmp    <=  #(TPROP_PCB_CTRL) ddr3_odt_fpga;
assign ddr3_odt_sdram =  ddr3_odt_sdram_tmp;

// Controlling the bi-directional BUS

genvar dqwd;
generate
    for (dqwd = 1; dqwd < DQ_WIDTH; dqwd = dqwd+1) begin : dq_delay
        WireDelay #
        (
            .Delay_g      (TPROP_PCB_DATA),
            .Delay_rd      (TPROP_PCB_DATA_RD),
            .ERR_INSERT    ("OFF")
        )
        u_delay_dq
        (
            .A              (ddr3_dq_fpga[dqwd]),
            .B              (ddr3_dq_sdram[dqwd]),

```

```

        .reset      (sys_rst_n),
        .phy_init_done (init_calib_complete)
    );
end

    WireDelay #
    (
        .Delay_g      (TPROP_PCB_DATA),
        .Delay_rd      (TPROP_PCB_DATA_RD),
        .ERR_INSERT ("OFF")
    )
    u_delay_dq_0
    (
        .A              (ddr3_dq_fpga[0]),
        .B              (ddr3_dq_sdram[0]),
        .reset          (sys_rst_n),
        .phy_init_done (init_calib_complete)
    );
endgenerate

genvar dqswd;
generate
    for (dqswd = 0; dqswd < DQS_WIDTH; dqswd = dqswd+1) begin : dqswd_delay
        WireDelay #
        (
            .Delay_g      (TPROP_DQS),
            .Delay_rd      (TPROP_DQS_RD),
            .ERR_INSERT ("OFF")
        )
        u_delay_dqs_p
        (
            .A              (ddr3_dqs_p_fpga[dqswd]),
            .B              (ddr3_dqs_p_sdram[dqswd]),
            .reset          (sys_rst_n),
            .phy_init_done (init_calib_complete)
        );

        WireDelay #
        (
            .Delay_g      (TPROP_DQS),
            .Delay_rd      (TPROP_DQS_RD),
            .ERR_INSERT ("OFF")
        )
        u_delay_dqs_n
        (
            .A              (ddr3_dqs_n_fpga[dqswd]),
            .B              (ddr3_dqs_n_sdram[dqswd]),
            .reset          (sys_rst_n),
            .phy_init_done (init_calib_complete)
        )
    end
endgenerate

```

```

    );
end
endgenerate

//=====
//                      FPGA Memory Controller
//=====

mig_ddr_test
mig_ddr_test_inst
(

    .ddr3_dq          (ddr3_dq_fpga),
    .ddr3_dqs_n       (ddr3_dqs_n_fpga),
    .ddr3_dqs_p       (ddr3_dqs_p_fpga),

    .ddr3_addr        (ddr3_addr_fpga),
    .ddr3_ba          (ddr3_ba_fpga),
    .ddr3_ras_n       (ddr3_ras_n_fpga),
    .ddr3_cas_n       (ddr3_cas_n_fpga),
    .ddr3_we_n        (ddr3_we_n_fpga),
    .ddr3_reset_n     (ddr3_reset_n),
    .ddr3_ck_p        (ddr3_ck_p_fpga),
    .ddr3_ck_n        (ddr3_ck_n_fpga),
    .ddr3_cke         (ddr3_cke_fpga),
    .ddr3_cs_n        (ddr3_cs_n_fpga),
    .ddr3_dm          (ddr3_dm_fpga),
    .ddr3_odt         (ddr3_odt_fpga),
    .sysclk_i         (sys_clk_i),
    .init_calib_complete (init_calib_complete),
    .tg_compare_error   (tg_compare_error)

);

//*****//
// Memory Models instantiations
//*****//

genvar r,i;
generate
for (r = 0; r < CS_WIDTH; r = r + 1) begin: mem_rnk
    if(DQ_WIDTH/16) begin: mem
        for (i = 0; i < NUM_COMP; i = i + 1) begin: gen_mem
            ddr3_model u_comp_ddr3
            (
                .rst_n    (ddr3_reset_n),
                .ck        (ddr3_ck_p_sdram),

```

```

        .ck_n      (ddr3_ck_n_sdrām),
        .cke       (ddr3_cke_sdrām[r]),
        .cs_n      (ddr3_cs_n_sdrām[r]),
        .ras_n     (ddr3_ras_n_sdrām),
        .cas_n     (ddr3_cas_n_sdrām),
        .we_n      (ddr3_we_n_sdrām),
        .dm_tdq̄s (ddr3_dm_sdrām[(2*(i+1)-1):(2*i)]),
        .ba        (ddr3_ba_sdrām[r]),
        .addr       (ddr3_addr_sdrām[r]),
        .dq         (ddr3_dq_sdrām[16*(i+1)-1:16*(i)]),
        .dq̄s       (ddr3_dq̄s_p_sdrām[(2*(i+1)-1):(2*i)]),
        .dq̄s_n     (ddr3_dq̄s_n_sdrām[(2*(i+1)-1):(2*i)]),
        .tdq̄s_n    (),
        .odt        (ddr3_odt_sdrām[r])
    );

end

end

if (DQ_WIDTH%16) begin: gen_mem_extrabits
    ddr3_model u_comp_ddr3
    (
        .rst_n      (ddr3_reset_n),
        .ck         (ddr3_ck_p_sdrām),
        .ck_n       (ddr3_ck_n_sdrām),
        .cke        (ddr3_cke_sdrām[r]),
        .cs_n       (ddr3_cs_n_sdrām[r]),
        .ras_n      (ddr3_ras_n_sdrām),
        .cas_n      (ddr3_cas_n_sdrām),
        .we_n       (ddr3_we_n_sdrām),
        .dm_tdq̄s ({ ddr3_dm_sdrām[DM_WIDTH-1], ddr3_dm_sdrām[DM_WIDTH-1]}),
        .ba         (ddr3_ba_sdrām[r]),
        .addr       (ddr3_addr_sdrām[r]),
        .dq         ({ ddr3_dq_sdrām[DQ_WIDTH-1:(DQ_WIDTH-8)],
                       ddr3_dq_sdrām[DQ_WIDTH-1:(DQ_WIDTH-8)]}),
        .dq̄s       ({ ddr3_dq̄s_p_sdrām[DQS_WIDTH-1],
                       ddr3_dq̄s_p_sdrām[DQS_WIDTH-1]}),
        .dq̄s_n     ({ ddr3_dq̄s_n_sdrām[DQS_WIDTH-1],
                       ddr3_dq̄s_n_sdrām[DQS_WIDTH-1]}),
        .tdq̄s_n    (),
        .odt        (ddr3_odt_sdrām[r])
    );

end

end

endgenerate

```

```

//*****

```



```
// Reporting the test case status
// Status reporting logic exists both in simulation test bench (sim_tb_top)
// and sim.do file for ModelSim. Any update in simulation run time or time out
// in this file need to be updated in sim.do file as well.
//*****

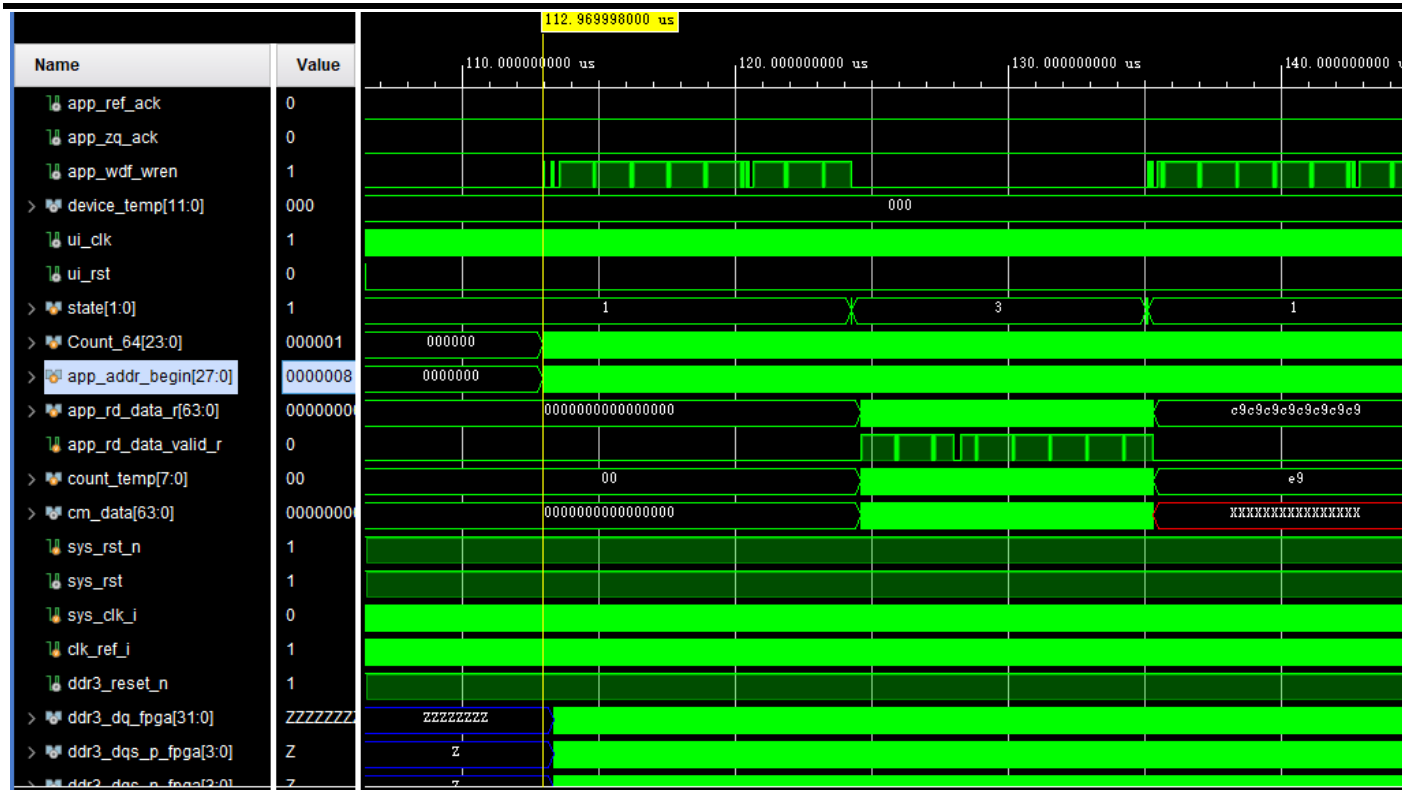
initial
begin : Logging
    fork
        begin : calibration_done
            wait (init_calib_complete);
            $display("Calibration Done");
            #500000000.0;
            if (!tg_compare_error) begin
                $display("TEST PASSED");
            end
            else begin
                $display("TEST FAILED: DATA ERROR");
            end
            disable calib_not_done;
            $finish;
        end

        begin : calib_not_done
            if (SIM_BYPASS_INIT_CAL == "SIM_INIT_CAL_FULL")
                #2500000000.0;
            else
                #1000000000.0;
            if (!init_calib_complete) begin
                $display("TEST FAILED: INITIALIZATION DID NOT COMPLETE");
            end
            disable calibration_done;
            $finish;
        end
    join
end

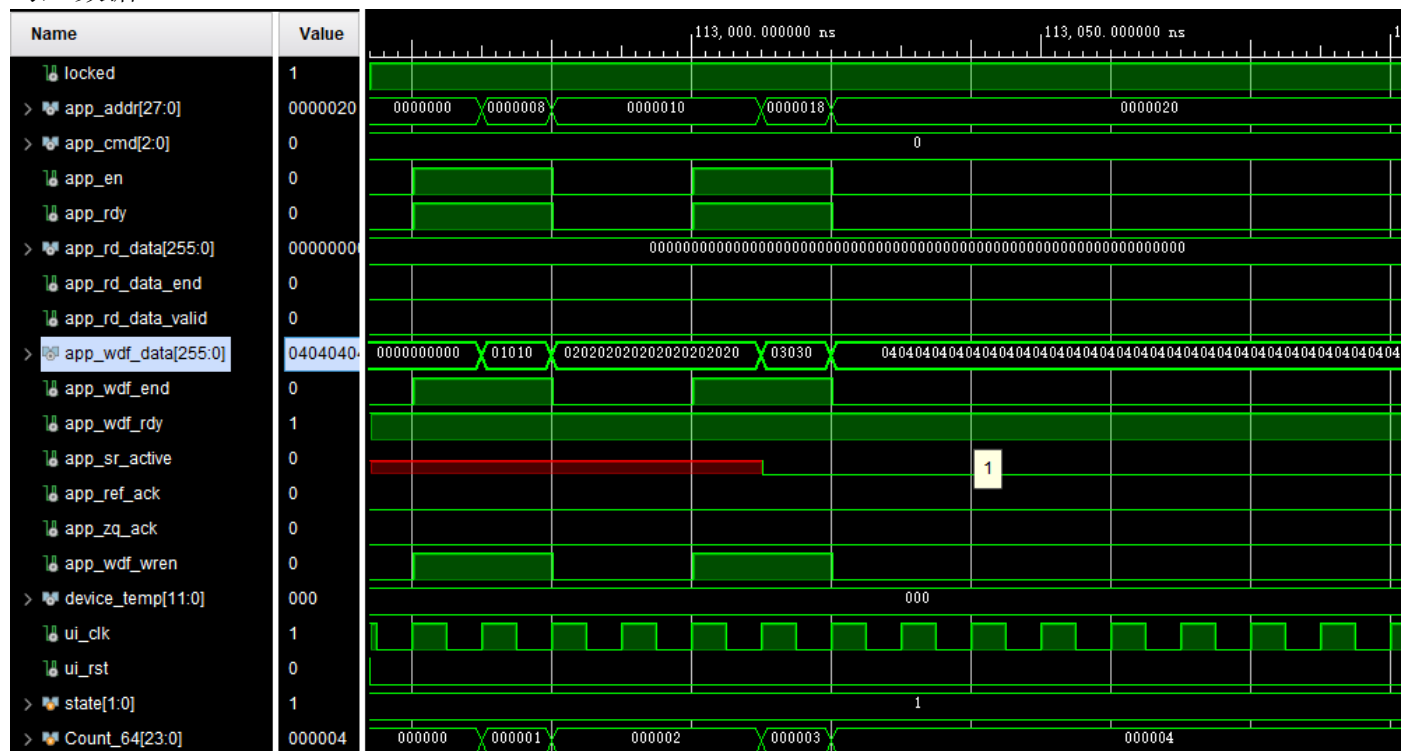
endmodule
```

## 1.7.4 仿真结果

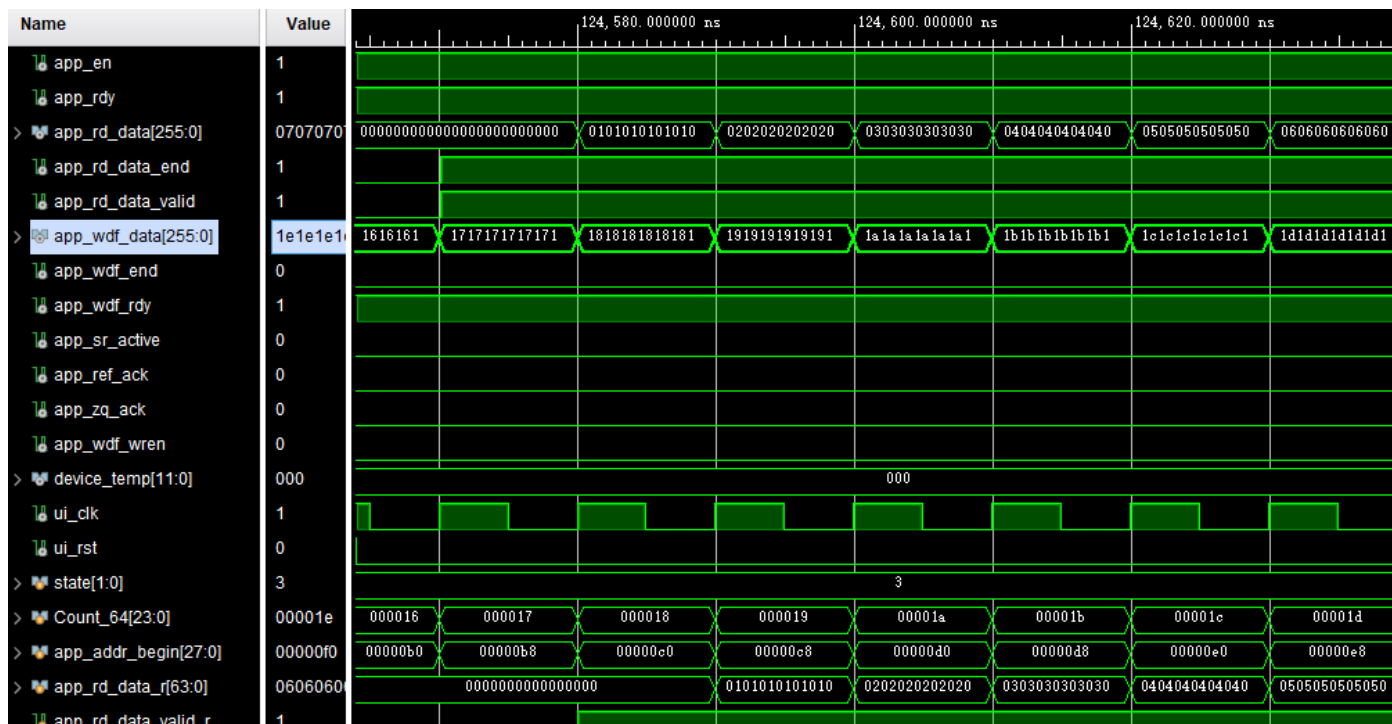
大概在 113us 左右，看到开始进行 MIG 写操作



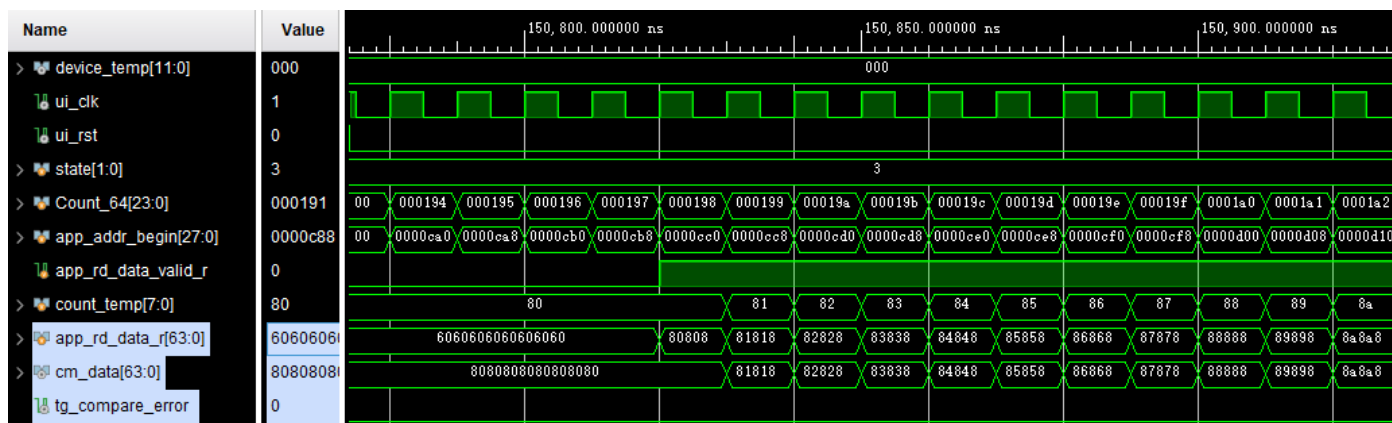
## 写入数据



## 读出数据



数据比对正确



## 1.8 VIVADO2019MIG 向导的 Bug

Vivado2019.2 mig 向导设置里面有个 bug，每次设置好后，重新打开向导设置，时钟输入默认会恢复到 400M 输入。整个读者需要注意，如果修改了 MIG 向导，时钟输入一定要重新设置。

## 02 基于 MIG 实现三缓存构架设计

软件版本: VIVADO2019.2

操作系统: WIN10 64bit

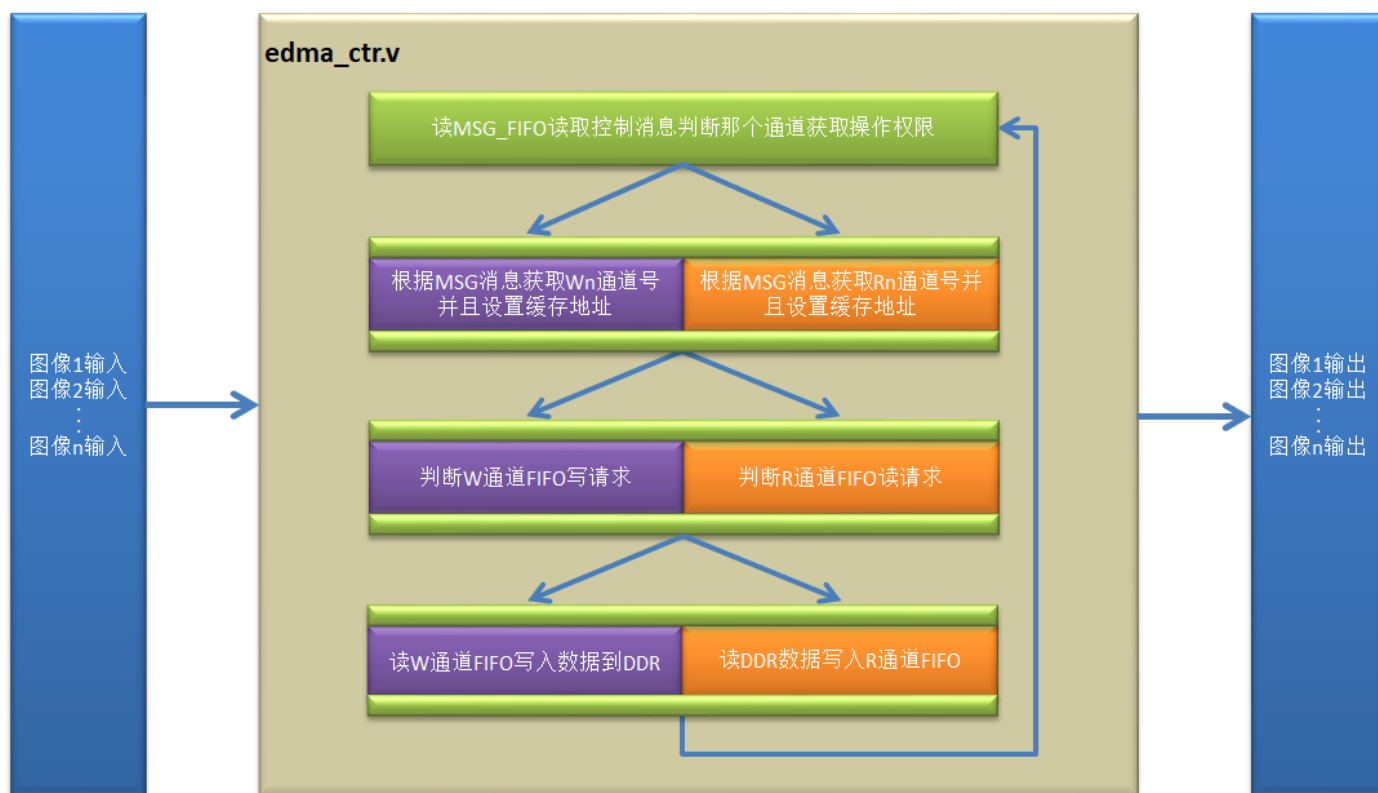
硬件平台: 适用 XILINX A7/K7/Z7/ZU/KU 系列 FPGA

登录米联客(MSXB0)FPGA 社区-[www.uisrc.com](http://www.uisrc.com) 观看免费视频课程、在线答疑解惑!

### 2.1 概述

优秀的程序都需要重视可重复性、可移植性、可维护性。米联客非常重视这一点，在新版本的demo在细节上再次改进。全方面提升代码的质量，每次做到“用心写好每一行代码”。在前面我们已经掌握了MIG的基本使用，那么本文中，我们就要设计一个优秀的代码模板，优秀的模板只要稍加修改就能用于一类应用。我们可以把这种模板成为代码的构架。优秀的代码从代码构架开始。

### 2.2 构架设计



先看上图中，我们设计的一种基于MIG实现的内存管理，支持多通道数据存储。具体内容在代码`edma_ctr.v`中实现。主要思路如下：

- 1)、MSG\_FIFO缓存每一帧的VS信号
- 2)、读取MSG\_FIFO中缓存的信号，判断获取读通道或者写通道的缓存地址
- 3)、判断W写通道数据FIFO写DDR请求或者R读通道数据FIFO读DDR请求
- 4)、把相应通道数据写入到DDR空间中或者从DDR空间读出

## 2.3edma\_ctr.v

测试样板以 MA703FA-100T 为例，如果是 MA703FA-35T MIG 数据位宽为 127Bit,具体可以参考配套代码。

```
`timescale 1ns / 1ps

/////////////////////////////////////////////////////////////////
/*
Company : Liyang Milian Electronic Technology Co., Ltd.
Brand: 米联客(msxbo)
Technical forum:uisrc.com
taobao: osrc.taobao.com
Create Date: 2019/02/27 22:09:55
Module Name: edma_ctr
Description: ddr image buf test
Copyright: Copyright (c) msxbo
Revision: 1.0
Signal description:?
1) _i input
2) _o output
3) _n activ low
4) _dg debug signal
5) _r delay or register
6) _s state mechine
*/

// W0_in_FIFO----->DDR----->R0_out_FIFO
/////////////////////////////////////////////////////////////////

module edma_ctr#
(
    parameter integer ADDR_OFFSET = 0,
    parameter integer BUF_SIZE = 3
)
(
    output [27:0] app_addr,
    output [2:0] app_cmd,
    output app_en,
    output [255:0] app_wdf_data,
    output app_wdf_end,
    output app_wdf_wren,
    input [255:0] app_rd_data,
    input app_rd_data_valid,
    input app_rdy,
    input app_wdf_rdy,
    input ui_clk,
    input ui_rstn_i,
    //Sensor video 640x480p -W0_FIFO
    input W0_fs_i,
    input W0_wclk_i,
    input W0_wren_i,
```

```

input      [31:0]      W0_data_i,//rgb565
//vga/hdmi output -R0_FIFO

input      R0_fs_i,
input      R0_rclk_i,
input      R0_rden_i,
output     [31:0]      R0_data_o
);

parameter PKG_SIZE      =8'd80;//一行像素
parameter RD_BURST_LEN  =8'd80;//一次 burst 80x256/32=640 个像素画 也就是一行数据
parameter WR_BURST_LEN  =8'd80;//一次 burst 80x256/32=640 个像素画 也就是一行数据

parameter [2:0]S_MFIFO0  =3'd0;
parameter [2:0]S_MFIFO1  =3'd1;
parameter [2:0]S_DATA0   =3'd3;
parameter [2:0]S_DATA1   =3'd4;
parameter [2:0]S_DATA2   =3'd5;
parameter [2:0]S_WDATA   =3'd6;
parameter [2:0]S_RDATA   =3'd7;
//----- 控制信息 FIFO 信号 -----//
reg        msg_wren=1'b0;
reg        msg_rden=1'b0;
reg  [7:0]msg_wdata=10'b0;
wire        msg_clk;
wire [7:0]msg_rdata;
wire        msg_full;
wire [4:0]msg_dcnt;
//----- W0 写通道 FIFO 信号 -----//

wire[255:0]W0_data_o;
wire[7:0]  W0_rcnt;
wire      W0_rclk_i;
//----- R0 读通道 FIFO 信号 -----//
wire[255:0]R0_data_i;
wire[7:0]  R0_rcnt;
wire      R0_wclk_i;

//----- FIFO 读写时钟处理 -----//
assign R0_wclk_i    =    ui_clk;
assign W0_rclk_i    =    ui_clk;
assign msg_clk      =    ui_clk;

wire  W0_fs_cap;
wire  R0_fs_cap;

fs_cap fs_cap_W0(.clk_i(ui_clk),.rstn_i(ui_rstn_i),.vs_i(W0_fs_i),.fs_cap_o(W0_fs_cap));
fs_cap fs_cap_R0(.clk_i(ui_clk),.rstn_i(ui_rstn_i),.vs_i(R0_fs_i),.fs_cap_o(R0_fs_cap));

```

```
//-----写控制信号进入 MSG_FIFO-----//
always@(posedge ui_clk)begin
    if(!ui_rstn_i)begin//--ddr 校准完成--//
        msg_wren    <=1'd0;
        msg_wdata    <=8'd0;
    end
    else begin
        msg_wren    <= W0_fs_cap || R0_fs_cap;
        msg_wdata    <={ W0_fs_cap,1'b0,1'b0,1'b0,R0_fs_cap,1'b0,1'b0,1'b0};
    end
end

reg W0_REQ = 0;
reg R0_REQ = 0;
always@(posedge ui_clk)begin
    W0_REQ    <= (W0_rcnt    > (PKG_SIZE-4));
    R0_REQ    <= (R0_rcnt    < (PKG_SIZE-4));
end

reg  W0_FIFO_Rst=1'b1;
reg  R0_FIFO_Rst=1'b1;
reg  [2:0]M_S = 3'd0;
reg  [7:0]rst_FIFO_cnt= 6'b0;
reg  [7:0]count_rden   = 8'd0;
reg  [7:0]count_wren   = 8'd0;

//-----地址空间-----//
reg [6:0]R0_Fbuf   = 7'd0;//缓存切换高地址
reg [6:0]W0_Fbuf   = 7'd0;//缓存切换高地址
reg [20:0]W0_addr   = 0;
reg [20:0]R0_addr   = 0;

//-----DDR 接口-----//
parameter [2:0]CMD_WRITE    =3'd0;//write cmd
parameter [2:0]CMD_READ     =3'd1;//read cmd

wire W0_rden_i;
wire R0_wren_i;

assign  app_wdf_end    = app_wdf_wren;//两个相等即可
assign  app_en         = (M_S==S_WDATA) ? (app_rdy&app_wdf_rdy) : ((M_S==S_RDATA)&app_rdy);//控制命令使能
assign  app_cmd        = (M_S==S_WDATA) ? CMD_WRITE : CMD_READ;//控制命令
assign  app_addr       = (M_S==S_WDATA) ? ({1'b0,W0_Fbuf,W0_addr}+ ADDR_OFFSET) : ({1'b0,R0_Fbuf,R0_addr}+ ADDR_OFFSET);//读写地址切换
assign  app_wdf_data    = W0_data_o;//写入的数据是计数器
assign  app_wdf_wren    = (M_S==S_WDATA)&app_rdy&app_wdf_rdy;//写使能
assign  W0_rden_i      = app_wdf_wren; //W0-FIFO 读使能
```

```

assign    R0_data_i    = app_rd_data; // R0-FIFO 写入的数据
assign    R0_wren_i     = app_rd_data_valid; // R0-FIFO 写使能

always@(posedge ui_clk)begin
    if(!ui_rstn_i)begin
        M_S            <= S_MFIFO0;
        msg_rden        <= 1'd0;
        W0_FIFO_Rst     <= 1'd0;
        R0_FIFO_Rst     <= 1'd0;
        rst_FIFO_cnt    <= 8'd0;
        count_rden      <= 8'd0;
        count_wren      <= 8'd0;
        W0_addr         <= 21'd0;
        R0_addr         <= 21'd0;
        W0_Fbuf         <= 7'd0;
        R0_Fbuf         <= 7'd0;
    end
    else case(M_S)
        //-----读取 FIFO 的控制信号-----//
        S_MFIFO0:begin//--FIFO 有数据就读取--//
            M_S            <=({msg_full,msg_dcnt}!=5'd0) ? S_MFIFO1 : S_DATA2;
            msg_rden        <=({msg_full,msg_dcnt}!=5'd0);
            W0_FIFO_Rst     <=1'd0;
            R0_FIFO_Rst     <=1'd0;
            rst_FIFO_cnt    <=8'd0;
            count_rden      <=8'd0;
            count_wren      <=8'd0;
        end
        S_MFIFO1:begin
            msg_rden        <= 1'b0;
            M_S            <= S_DATA0;
        end
        //-----读取 FIFO 的控制信号-----//
        S_DATA0:begin//--相对地址处理 多缓存切换--//
            M_S            <= S_DATA1;
            if(msg_rdata[7])begin//wirte
                W0_addr <= 21'd0;
                if(W0_Fbuf == (BUF_SIZE-1))
                    W0_Fbuf <= 0;
                else
                    W0_Fbuf <= W0_Fbuf + 1'b1;
            end
            if(msg_rdata[3])begin//read
                R0_addr <= 21'd0;
                if(W0_Fbuf == 0)
                    R0_Fbuf <= (BUF_SIZE-1);
                else
                    R0_Fbuf <= W0_Fbuf - 1'b1;
            end
        end
    endcase
end

```



```

        end
    end
    S_DATA1:begin
        M_S          <= (rst_FIFO_cnt>=8'd60) ? S_DATA2 : M_S;
        R0_FIFO_Rst   <= (rst_FIFO_cnt<=8'd20)&&msg_rdata[3];
        W0_FIFO_Rst   <= (rst_FIFO_cnt<=8'd20)&&msg_rdata[7];
        rst_FIFO_cnt   <= rst_FIFO_cnt + 8'd1;
    end
    //-----状态机空闲状态-----//
    S_DATA2:begin
        count_rden     <=8'd0;
        count_wren     <=8'd0;
        casex({ W0_REQ&1'b1,1'b0,1'b0,R0_REQ&1'b1,1'b0,1'b0,1'b0})
            8'b1???_????: M_S   <= S_WDATA; //ddr 写数据-//
            8'b0000_1000: M_S   <= S_RDATA; //ddr 读数据-//
            default:      M_S   <= S_MFIFO0;
        endcase
    end
    //-----sdram 读状态-----//
    S_RDATA:begin //--读取数据--//
        R0_addr        <= app_rdy ? (R0_addr+4'd8) : R0_addr ;//每次写入的数据地址
        M_S             <= app_rdy&&(count_rden==RD_BURST_LEN-1'b1) ? S_MFIFO0:M_S; //释放本次写权限
        count_rden      <= app_rdy ? count_rden+1'd1:count_rden;//count_rden 用来标记一次 burst 的数据量
    end
    //-----sdram 写状态-----//
    S_WDATA:begin//--写入数据--//
        W0_addr        <= app_rdy&&app_wdf_rdy ? (W0_addr+4'd8) : W0_addr ;//每次写入的数据地址
        M_S             <= app_rdy&&app_wdf_rdy&&(count_wren==WR_BURST_LEN-1'b1) ? S_MFIFO0:M_S;//释放本次写
        count_wren      <= app_rdy&&app_wdf_rdy ? count_wren+1'd1:count_wren;//count_wren 用来标记一次 burst 的数据量
    end
    default:begin
        M_S             <=S_MFIFO0;
        count_rden      <=8'd0;
        count_wren      <=8'd0;
    end
endcase
end
//-----控制消息 fifo 接口-----//
ms_fifo ms_fifo_inst (
    .clk(msg_clk),      // input wire clk
    .din(msg_wdata),    // input wire [9 : 0] din
    .wr_en(msg_wren),   // input wire wr_en
    .rd_en(msg_rden),   // input wire rd_en
    .dout(msg_rdata),   // output wire [9 : 0] dout
    .full(msg_full),    // output wire full
    .empty(empty),      // output wire empty
    .data_count(msg_dcnt) // output wire [4 : 0] data_count

```

```
);
//-----Sensor fifo 接口-----//
wr_fifo wr_fifo_inst (
    .rst(W0_FIFO_Rst), // input wire rst
    .wr_clk(W0_wclk_i), // input wire wr_clk
    .rd_clk(W0_rclk_i), // input wire rd_clk
    .din(W0_data_i), // input wire [31 : 0] din
    .wr_en(W0_wren_i), // input wire wr_en
    .rd_en(W0_rden_i), // input wire rd_en
    .dout(W0_data_o), // output wire [255 : 0] dout
    .rd_data_count(W0_rcnt) // output wire [7 : 0] rd_data_count
);
//-----R0 fifo 接口-----//
rd_fifo rd_fifo_inst (
    .rst(R0_FIFO_Rst), // input wire rst
    .wr_clk(R0_wclk_i), // input wire wr_clk
    .rd_clk(R0_rclk_i), // input wire rd_clk
    .din(R0_data_i), // input wire [255 : 0] din
    .wr_en(R0_wren_i), // input wire wr_en
    .rd_en(R0_rden_i), // input wire rd_en
    .dout(R0_data_o), // output wire [31 : 0] dout
    .wr_data_count(R0_rcnt) // output wire [7 : 0] wr_data_count
);

//以下是 debug 信号
/*
ila_0 debug_inst0 (
    .clk(ui_clk), // input wire clk
    .probe0(W0_data_o), // input wire [255:0] probe0
    .probe1(R0_data_i), // input wire [31:0] probe1
    .probe2(R0_wren_i),
    .probe3(app_en),
    .probe4(W0_rcnt[6:0]),
    .probe5(app_wdf_rdy),
    .probe6(app_wdf_wren),
    .probe7(count_wren[6:0]),
    .probe8(app_rdy),
    .probe9(count_rden[6:0]),
    .probe10(M_S)
);

ila_1 debug_inst1 (
    .clk(R0_rclk_i), // input wire clk
    .probe0(R0_data_o), // input wire [31:0] probe0
    .probe1(R0_rden_i), // input wire [0:0] probe1
    .probe2(R0_fs_i) // input wire [0:0] probe1
);
*/
```

endmodule

本代码演示了消息支持 8 路消息，ms\_fifo 为 8 路位宽，缓存深度为 32 深度的消息机制。一路写入通道，一路读出通道，写入和输出采用三缓存设计。适合图像类缓存。并且代码稍加修改可以增加至 8 个通道的写入或者读出。

### 2.3.1 ms\_fifo ip 设置

Component Name: ms\_fifo

**Basic** Native Ports Status Flags Data Counts Summary

**Interface Type**

☒ Native ☐ AXI Memory Mapped ☐ AXI Stream

Fifo Implementation: Common Clock Block RAM

**FIFO Implementation Options**

Supported Features

	Memory Type	(1)	(2)	(3)	(4)	(5)
Common Clock (CLK)	Block RAM	✓	✓		✓	✓
Common Clock (CLK)	Distributed RAM		✓			
Common Clock (CLK)	Shift Register					
Common Clock (CLK)	Built-in FIFO		✓	✓	✓	✓
Independent Clocks (RD_CLK, WR_CLK)	Block RAM	✓	✓		✓	✓
Independent Clocks (RD_CLK, WR_CLK)	Distributed RAM		✓			
Independent Clocks (RD_CLK, WR_CLK)	Built-in FIFO		✓	✓	✓	✓

(1) Non-symmetric aspect ratios (different read and write data widths)  
 (2) First-Word Fall-Through  
 (3) Uses Built-in FIFO primitives  
 (4) ECC support  
 (5) Dynamic Error Injection

Component Name: ms\_fifo

**Basic** Native Ports Status Flags Data Counts Summary

**Read Mode**

☒ Standard FIFO ☐ First Word Fall Through

**Data Port Parameters**

Write Width: 8 (1,2,3,...,1024)

Write Depth: 32 (Actual Write Depth: 32)

Read Width: 8

Read Depth: 32 (Actual Read Depth: 32)

**ECC, Output Register and Power Gating Options**

☐ ECC ☒ Hard ECC ☐ Single Bit Error Injection ☐ Double Bit Error Injection

☐ ECC Pipeline Reg ☐ Dynamic Power Gating

☐ Output Registers ☒ Embedded Registers

**Initialization**

☐ Reset Pin ☒ Enable Reset Synchronization ☐ Enable Safety Circuit

Reset Type: Asynchronous Reset

Full Flags Reset Value: 0

☐ Dout Reset Value: Previous dout Value

Read Latency: 1

Component Name

Basic Native Ports Status Flags **Data Counts** Summary

**Data Count Options**

☐ More Accurate Data Counts

☒ Data Count

Data Count Width  [1 - 5]

☐ Write Data Count (Synchronized with Write Clk)

Write Data Count Width  [1 - 5]

☐ Read Data Count (Synchronized with Read Clk)

Read Data Count Width  [1 - 5]

## 2.3.2 wr\_fifo ip 设置

Component Name

Basic Native Ports Status Flags **Data Counts** Summary

**Interface Type**

☒ Native ☐ AXI Memory Mapped ☐ AXI Stream

Fifo Implementation

Synchronization Stages

**FIFO Implementation Options**

Supported Features

	Memory Type	(1)	(2)	(3)	(4)	(5)
Common Clock (CLK)	Block RAM	✓	✓		✓	✓
Common Clock (CLK)	Distributed RAM		✓			
Common Clock (CLK)	Shift Register					
Common Clock (CLK)	Built-in FIFO		✓	✓	✓	✓
<b>Independent Clocks (RD_CLK, WR_CLK)</b>	<b>Block RAM</b>	✓	✓		✓	✓
Independent Clocks (RD_CLK, WR_CLK)	Distributed RAM		✓			
Independent Clocks (RD_CLK, WR_CLK)	Built-in FIFO		✓	✓	✓	✓

(1) Non-symmetric aspect ratios (different read and write data widths)  
 (2) First-Word Fall-Through  
 (3) Uses Built-in FIFO primitives  
 (4) ECC support  
 (5) Dynamic Error Injection

测试样板以 MA703FA-100T 为例，如果是 MA703FA-35T MIG 数据位宽为 127Bit,具体可以参考配套代码。

Component Name

Basic

Native Ports

Status Flags

Data Counts

Summary

Read Mode

☐ Standard FIFO ☒ First Word Fall Through

Data Port Parameters

Write Width  1,2,3...1024

Write Depth  Actual Write Depth: 2063

Read Width

Read Depth  Actual Read Depth: 257

ECC, Output Register and Power Gating Options

☐ ECC  ☐ Single Bit Error Injection ☐ Double Bit Error Injection

☐ ECC Pipeline Reg ☐ Dynamic Power Gating

☐ Output Registers

Initialization

☒ Reset Pin ☒ Enable Reset Synchronization ☐ Enable Safety Circuit

Reset Type

Full Flags Reset Value

☒ Dout Reset Value  (Hex)

Read Latency : 0

Component Name

Basic

Native Ports

Status Flags

Data Counts

Summary

Data Count Options

☐ More Accurate Data Counts

☐ Data Count

Data Count Width  [1 - 11]

☐ Write Data Count (Synchronized with Write Clk)

Write Data Count Width  [1 - 11]

☒ Read Data Count (Synchronized with Read Clk)

Read Data Count Width  [1 - 8]

## 2.3.3rd\_fifo ip 设置

Component Name

**Basic** Native Ports Status Flags Data Counts Summary

**Interface Type**

☒ Native ☐ AXI Memory Mapped ☐ AXI Stream

Fifo Implementation

Synchronization Stages

**FIFO Implementation Options**

Supported Features

	Memory Type	(1)	(2)	(3)	(4)	(5)
Common Clock (CLK)	Block RAM	✓	✓		✓	✓
Common Clock (CLK)	Distributed RAM		✓			
Common Clock (CLK)	Shift Register					
Common Clock (CLK)	Built-in FIFO		✓	✓	✓	✓
<b>Independent Clocks (RD_CLK, WR_CLK)</b>	<b>Block RAM</b>	✓	✓		✓	✓
Independent Clocks (RD_CLK, WR_CLK)	Distributed RAM		✓			
Independent Clocks (RD_CLK, WR_CLK)	Built-in FIFO		✓	✓	✓	✓

(1) Non-symmetric aspect ratios (different read and write data widths)  
 (2) First-Word Fall-Through  
 (3) Uses Built-in FIFO primitives  
 (4) ECC support  
 (5) Dynamic Error Injection

测试样板以 MA703FA-100T 为例，如果是 MA703FA-35T MIG 数据位宽为 127Bit,具体可以参考配套代码。

Component Name

**Basic** **Native Ports** Status Flags Data Counts Summary

**Read Mode**

☐ Standard FIFO ☒ First Word Fall Through

**Data Port Parameters**

Write Width  1,2,3,..1024

Write Depth  Actual Write Depth: 255

Read Width

Read Depth  Actual Read Depth: 2040

**ECC, Output Register and Power Gating Options**

☐ ECC  ☐ Single Bit Error Injection ☐ Double Bit Error Injection

☐ ECC Pipeline Reg ☐ Dynamic Power Gating

☐ Output Registers

**Initialization**

☒ Reset Pin ☒ Enable Reset Synchronization ☐ Enable Safety Circuit

Reset Type

Full Flags Reset Value

☒ Dout Reset Value  (Hex)

Read Latency : 0

Component Name **rd\_fifo**

Basic	Native Ports	Status Flags	Data Counts	Summary
<b>Data Count Options</b>				
<input type="checkbox"/> More Accurate Data Counts				
<input type="checkbox"/> Data Count				
Data Count Width <input type="text" value="8"/> [1 - 8]				
<input checked="" type="checkbox"/> Write Data Count (Synchronized with Write Clk)				
Write Data Count Width <input type="text" value="8"/> [1 - 8]				
<input type="checkbox"/> Read Data Count (Synchronized with Read Clk)				
Read Data Count Width <input type="text" value="11"/> [1 - 11]				

## 2.3.4 帧信号写入消息 FIFO

```
//-----写控制信号进入MSG_FIFO-----//
always@(posedge ui_clk)begin
    if(!ui_rstn_i)begin//--ddr校准完成--//
        msg_wren    <=1'd0;
        msg_wdata    <=8'd0;
    end
    else begin
        msg_wren    <= W0_fs_cap || R0_fs_cap;
        msg_wdata    <={W0_fs_cap, 1'b0, 1'b0, 1'b0, R0_fs_cap, 1'b0, 1'b0, 1'b0};
    end
end
end
```

### fs\_cap.v

```
`timescale 1ns / 1ps
/*
Company : Liyang Milian Electronic Technology Co., Ltd.
Brand: 米联客(msxbo)
Technical forum:uisrc.com
taobao: osrc.taobao.com
Create Date: 2019/12/17
Module Name: fs_cap
Description:
Copyright: Copyright (c) msxbo
Revision: 1.0
Signal description:
1) _i input
2) _o output
3) _n activ low
4) _dg debug signal
5) _r delay or register
6) _s state mechine
```

```
*/  
/////////////////////////////////////  
  
module fs_cap(  
    input  clk_i,  
    input  rstn_i,  
    input  vs_i,  
    output reg fs_cap_o  
);  
  
//----CH0_CNT_FS 信号电平计数 实际就是采样 VS 信号-----  
reg[4:0]CNT_FS    = 6'b0;  
reg[4:0]CNT_FS_n = 6'b0;  
reg      FS       = 1'b0;  
(* ASYNC_REG = "TRUE" *) reg vs_i_r1;  
(* ASYNC_REG = "TRUE" *) reg vs_i_r2;  
(* ASYNC_REG = "TRUE" *) reg vs_i_r3;  
(* ASYNC_REG = "TRUE" *) reg vs_i_r4;  
//----同步整形电路，之前大意没加整形电路导致总是采集 vs 出错-----  
    always@(posedge clk_i) begin  
        vs_i_r1 <= vs_i;  
        vs_i_r2 <= vs_i_r1;  
        vs_i_r3 <= vs_i_r2;  
        vs_i_r4 <= vs_i_r3;  
    end  
  
    always@(posedge clk_i) begin  
        if(!rstn_i)begin  
            fs_cap_o <= 1'd0;  
        end  
        else if({vs_i_r4,vs_i_r3} == 2'b01)begin  
            fs_cap_o <= 1'b1;  
        end  
        else begin  
            fs_cap_o <= 1'b0;  
        end  
    end  
  
endmodule
```

### 2.3.5M\_S 内存管理状态机

```
assign  app_wdf_end  = app_wdf_wren;//两个相等即可  
assign  app_en       = (M_S==S_WDATA) ? (app_rdy&app_wdf_rdy) : ((M_S==S_RDATA)&app_rdy);//控制命令使能  
assign  app_cmd      = (M_S==S_WDATA) ? CMD_WRITE : CMD_READ;//控制命令  
assign  app_addr     = (M_S==S_WDATA) ? ({1'b0, W0_Fbuf, W0_addr}+ ADDR_OFFSET) : ({1'b0, R0_Fbuf, R0_addr}+ ADDR_OFFSET);//读写地址切换
```



```

assign    app_wdf_data  = W0_data_o;//写入的数据是计数器
assign    app_wdf_wren  = (M_S==S_WDATA)&app_rdy&app_wdf_rdy;//写使能
assign    W0_rden_i     = app_wdf_wren; //W0-FIFO 读使能
assign    R0_data_i     = app_rd_data; // R0-FIFO 写入的数据
assign    R0_wren_i     = app_rd_data_valid;//R0-FIFO 写使能

always@(posedge ui_clk)begin
    if(!ui_rstn_i)begin
        M_S            <= S_MFIFO0;
        msg_rden       <= 1'd0;
        W0_FIFO_Rst    <= 1'd0;
        R0_FIFO_Rst    <= 1'd0;
        rst_FIFO_cnt   <= 8'd0;
        count_rden     <= 8'd0;
        count_wren     <= 8'd0;
        W0_addr        <= 21'd0;
        R0_addr        <= 21'd0;
        W0_Fbuf        <= 7'd0;
        R0_Fbuf        <= 7'd0;
    end
    else case(M_S)
        //-----读取 FIFO 的控制信号-----//
        S_MFIFO0:begin//--FIFO 有数据就读取--//
            M_S            <=({msg_full,msg_dcnt}!=5'd0) ? S_MFIFO1 : S_DATA2;
            msg_rden       <=({msg_full,msg_dcnt}!=5'd0);
            W0_FIFO_Rst    <=1'd0;
            R0_FIFO_Rst    <=1'd0;
            rst_FIFO_cnt   <=8'd0;
            count_rden     <=8'd0;
            count_wren     <=8'd0;
        end
        S_MFIFO1:begin
            msg_rden       <= 1'b0;
            M_S            <= S_DATA0;
        end
        //-----读取 FIFO 的控制信号-----//
        S_DATA0:begin//--相对地址处理 多缓存切换--//
            M_S            <= S_DATA1;
            if(msg_rdata[7])begin//write
                W0_addr <= 21'd0;
                if(W0_Fbuf == (BUF_SIZE-1))
                    W0_Fbuf <= 0;
                else
                    W0_Fbuf <= W0_Fbuf + 1'b1;
            end
            if(msg_rdata[3])begin//read
                R0_addr <= 21'd0;
                if(W0_Fbuf == 0)

```

```

        RO_Fbuf <= (BUF_SIZE-1);
    else
        RO_Fbuf <= WO_Fbuf - 1'b1;
    end
end
S_DATA1:begin
    M_S          <= (rst_FIFO_cnt>=8'd60) ? S_DATA2 : M_S;
    RO_FIFO_Rst  <= (rst_FIFO_cnt<=8'd20)&&msg_rdata[3];
    WO_FIFO_Rst  <= (rst_FIFO_cnt<=8'd20)&&msg_rdata[7];
    rst_FIFO_cnt <= rst_FIFO_cnt + 8'd1;
end
//-----状态机空闲状态-----//
S_DATA2:begin
    count_rden    <=8'd0;
    count_wren    <=8'd0;
    casex({WO_REQ&1'b1, 1'b0, 1'b0, 1'b0, RO_REQ&1'b1, 1'b0, 1'b0, 1'b0})
        8'b1???_??? : M_S <= S_WDATA; //ddr 写数据--//
        8'b0000_1000 : M_S <= S_RDATA; //ddr 读数据--//
        default:      M_S <= S_MFIF00;
    endcase
end
//-----sdram 读状态-----//
S_RDATA:begin //--读取数据--//
    RO_addr      <= app_rdy ? (RO_addr+4'd8) : RO_addr ;//每次写入的数据地址
    M_S          <= app_rdy&&(count_rden==RD_BURST_LEN-1'b1) ? S_MFIF00:M_S; //释放本次写权限
    count_rden    <= app_rdy ? count_rden+1'd1:count_rden;//count_rden 用来标记一次 burst 的数据量
end
//-----sdram 写状态-----//
S_WDATA:begin//--写入数据--//
    WO_addr      <= app_rdy&&app_wdf_rdy ? (WO_addr+4'd8) : WO_addr ;//每次写入的数据地址
    M_S          <= app_rdy&&app_wdf_rdy&&(count_wren==WR_BURST_LEN-1'b1) ? S_MFIF00:M_S; //释放本次写权限
    count_wren    <= app_rdy&&app_wdf_rdy ? count_wren+1'd1:count_wren;//count_wren 用来标记一次 burst 的数据量
end
default:begin
    M_S          <=S_MFIF00;
    count_rden    <=8'd0;
    count_wren    <=8'd0;
end
endcase
end
end

```

## 2.3.6 读写 DDR 请求控制

```

always@(posedge ui_clk)begin
    WO_REQ    <= (WO_rcnt    > (PKG_SIZE-4));

```

```
R0_REQ    <= (R0_rcnt    < (PKG_SIZE-4));  
end
```

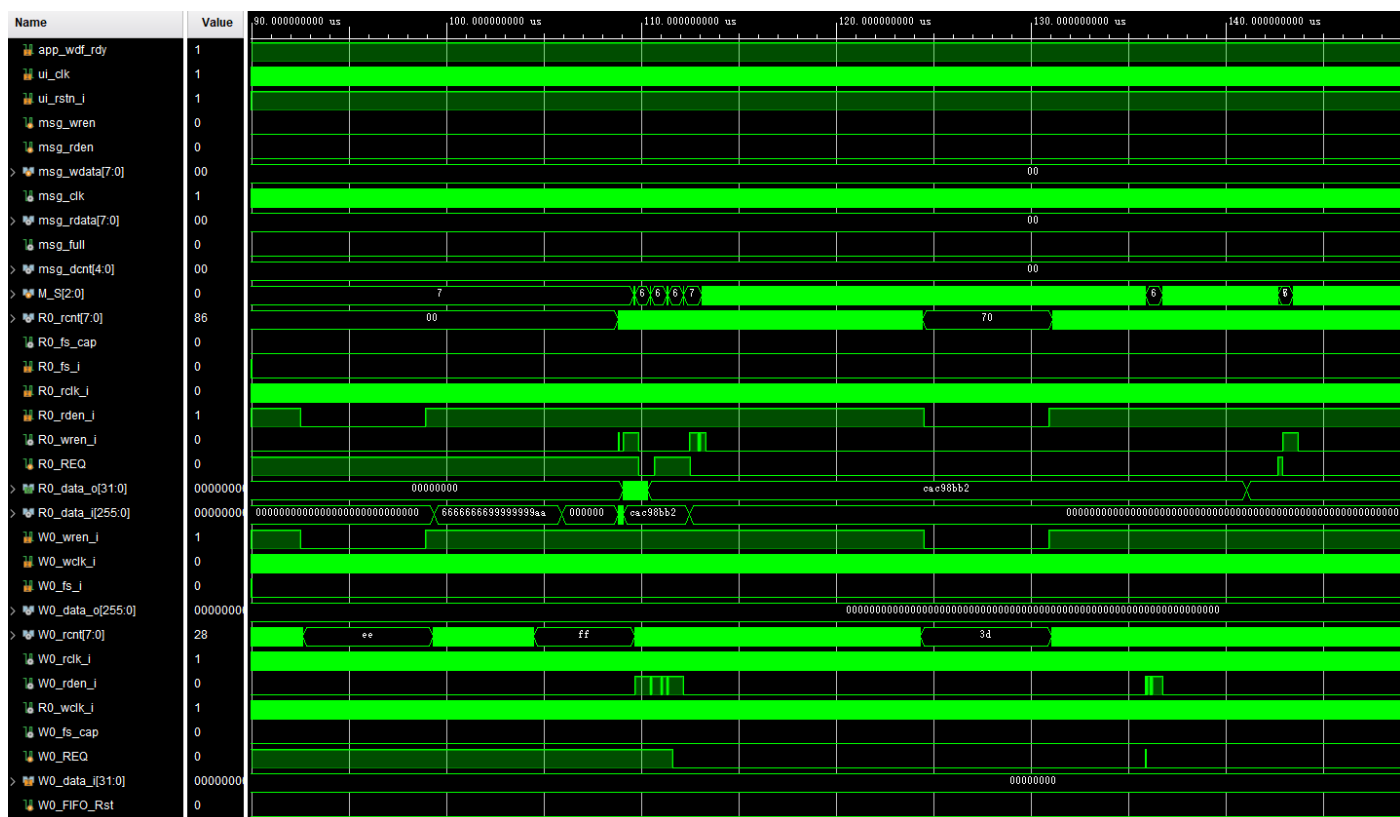
## 2.4 代码结构

```
▼ img_3buf_test (img_3buf_test.v) (7)  
  > clk_wiz0_inst: clk_wiz_0 (clk_wiz_0.xci)  
  > clk_wiz1_inst: clk_wiz_1 (clk_wiz_1.xci)  
  > u_mig_7series_0: mig_7series_0 (mig_7series_0.xci)  
  > uihdmity_inst: uihdmity (uihdmity.v) (8)  
  > uivtc_inst: uivtc (uivtc.v)  
  > uitpg_inst: uitpg (uitpg.v)  
  ▼ edma_ctr_inst: edma_ctr (edma_ctr.v) (5)  
    > fs_cap_W0: fs_cap (fs_cap.v)  
    > fs_cap_R0: fs_cap (fs_cap.v)  
    > ms_fifo_inst: ms_fifo (ms_fifo.xci)  
    > wr_fifo_inst: wr_fifo (wr_fifo.xci)  
    > rd_fifo_inst: rd_fifo (rd_fifo.xci)
```

以上代码中 uivtc ip 用来产生 RGB 时序，uitpg ip 用来产生测试图形，uihdmity ip 用来输出测试结果到 HDMI 显示器。

以上 IP 源码全部在“米联客(MSXBO)2020 版 FPGA 课程”中出现，不再重复。

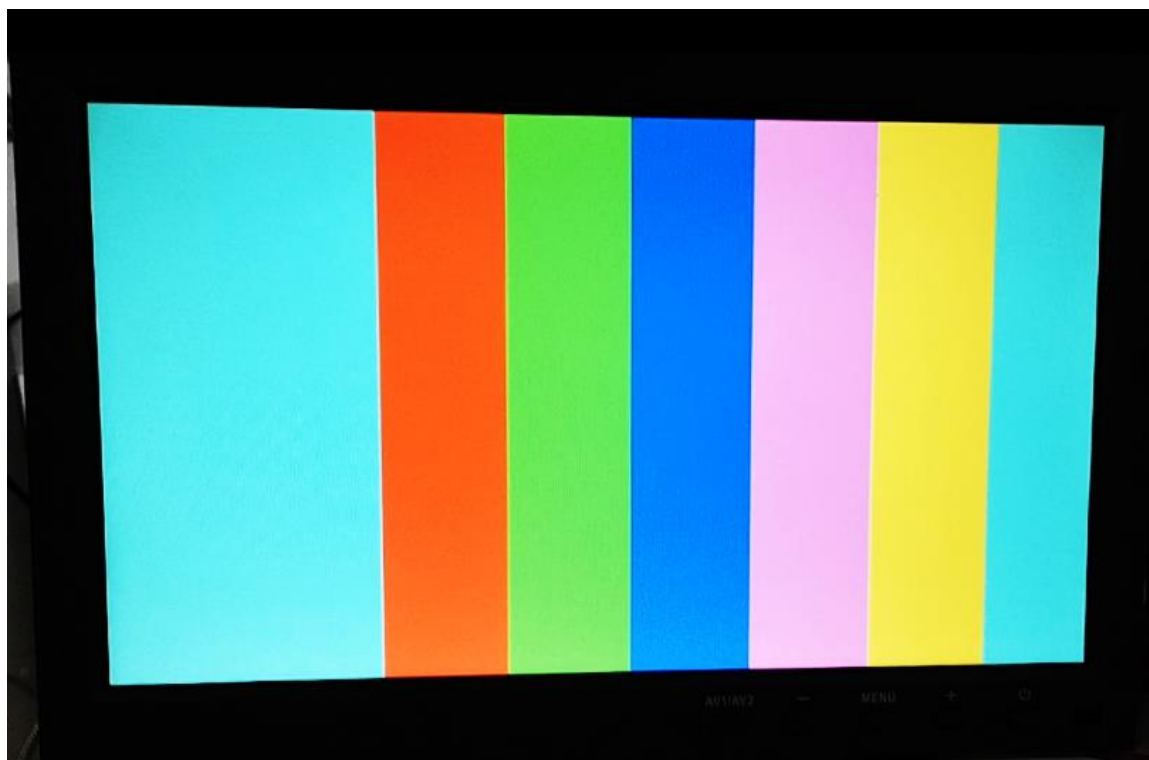
## 2.5RTL 仿真结果



如上图所示为波形仿真截图，包含了 MIG 内存控制器、wr\_fifo、rd\_fifo、ms\_fifo、M\_S 状态机的相关信号。程序的逻辑关系可以在此图上找到对应关系。详细的分析读者可以自己结合代码分析。

由于仿真速度非常慢，需要看到完整一帧图形的仿真过程，需要仿真一天时间。

## 2.6 输出测试图形



## 03 摄像头采集 DDR 三缓存

软件版本: VIVADO2019.2

操作系统: WIN10 64bit

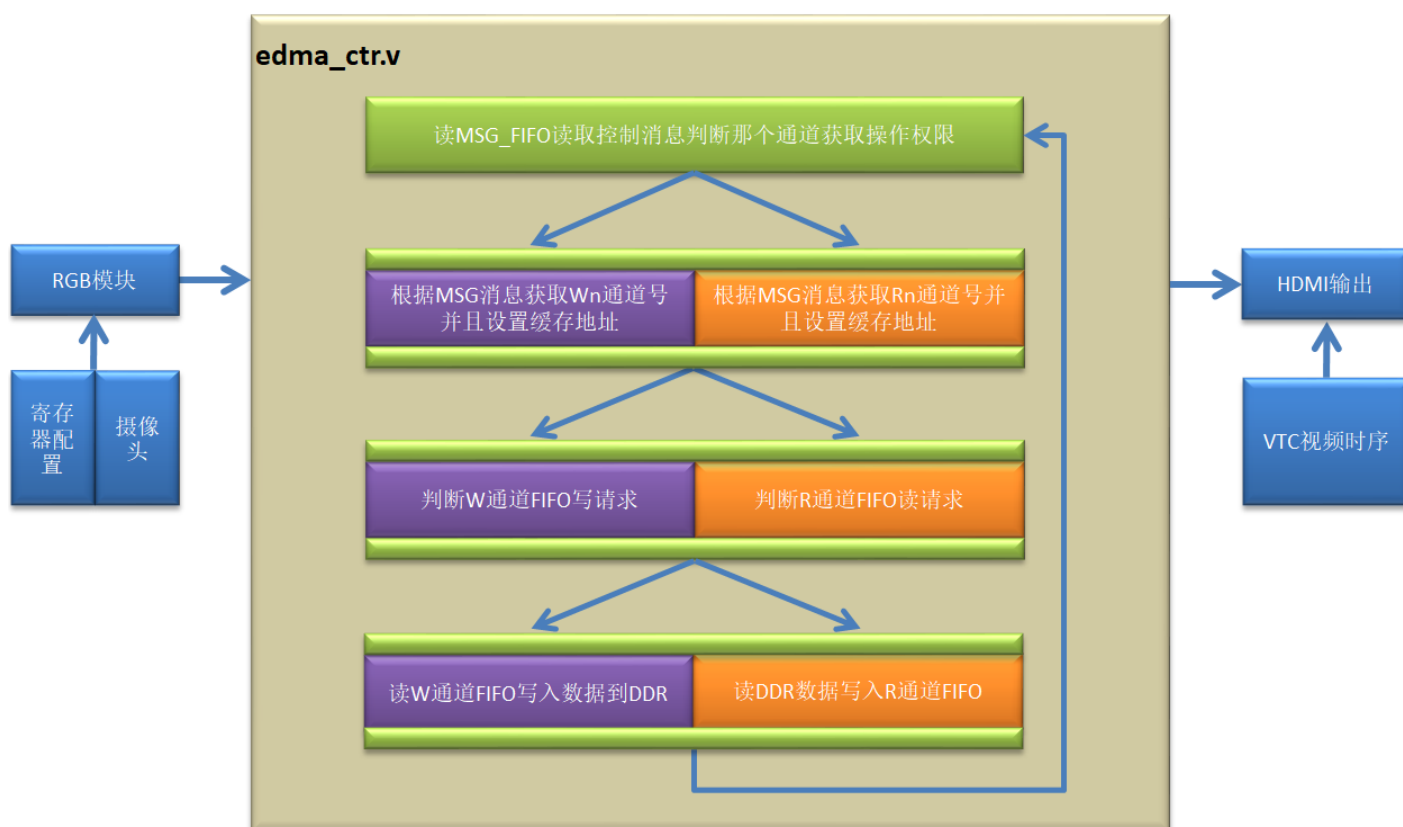
硬件平台: 适用 XILINX A7/K7/Z7/ZU/KU 系列 FPGA

登录米联客(MSXBO)FPGA 社区-[www.uisrc.com](http://www.uisrc.com) 观看免费视频课程、在线答疑解惑!

### 3.1 概述

基于前文的构架, 本文只要将前文代码稍加修改就能实现摄像头数据的缓存, 本文测试提供的测试 demo 包含了常用的 OV5640 摄像头、OV7725 摄像头、MT9V034 摄像头。由于摄像头都是 DVP 并口传输, 而且代码和方法基本一致。本文介绍以 OV7725 摄像头为例。

### 3.2 构架设计



对于核心控制模块 `edma_ctr.v` 几乎不用改变任何代码。只有 OV5640 分辨率是 1280\*720 需要修改下每次写入 DDR 的数量。

640\*480 或者 1280\*720 分辨率参数设置:

parameter PKG\_SIZE = 8'd80; // 对于 640\*480 一行像素; 对于 1280\*720 半行数据

parameter RD\_BURST\_LEN = 8'd80; // 一次 burst 80x256/32=640 个像素画

parameter WR\_BURST\_LEN = 8'd80; // 一次burst 80x256/32=640 个像素画

如果 1280\*720 分辨率参数也是采用以上值, 则每一行需要 burst 2 次。

1280\*720 分辨率参数最大值设置:

parameter PKG\_SIZE = 8'd160; // 一行像素

parameter RD\_BURST\_LEN = 8'd160; // 一次 burst 160x256/32=1280 个像素画 也就是一行数据

parameter WR\_BURST\_LEN = 8'd160; // 一次burst 160x256/32=1280 个像素画 也就是一行数据

这样只要 burst 1 次。

### 3.3 代码结构

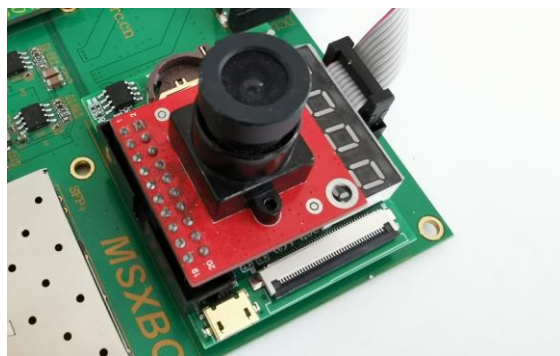
```
▼ ● img_3buf_ov7725 (img_3buf_test.v) (8)
  > 📁 clk_wiz0_inst: clk_wiz_0 (clk_wiz_0.xci)
  > 📁 clk_wiz1_inst: clk_wiz_1 (clk_wiz_1.xci)
  > 📁 u_mig_7series_0: mig_7series_0 (mig_7series_0.xci)
  > ● uihdmix_inst: uihdmix (uihdmix.v) (8)
  ● uivtc_inst: uivtc (uivtc.v)
  ▼ ● uicfg7725_inst: uicfg7725 (uicfg7725.v) (2)
    ● uii2c_inst: uii2c (uii2c.v)
    ● ui7725reg_inst: ui7725reg (ui7725reg.v)
    ● uiSensorRGB565_inst: uiSensorRGB565 (uiSensorRGB565.v)
  ▼ ● edma_ctr_inst: edma_ctr (edma_ctr.v) (5)
    ● fs_cap_W0: fs_cap (fs_cap.v)
    ● fs_cap_R0: fs_cap (fs_cap.v)
    > 📁 ms_fifo_inst: ms_fifo (ms_fifo.xci)
    > 📁 wr_fifo_inst: wr_fifo (wr_fifo.xci)
    > 📁 rd_fifo_inst: rd_fifo (rd_fifo.xci)
```

我们主要关注下红色线框内的代码

- 1)、uicfg7725.v ui7725reg.v uii2c.v负责对OV7725摄像头寄存器初始化，设置输出分辨率是640\*480@60fps. 输出的视频格式为RGB565
- 2)、uiSensorRGB565.v负责把RGB565转为RGB888数据格式

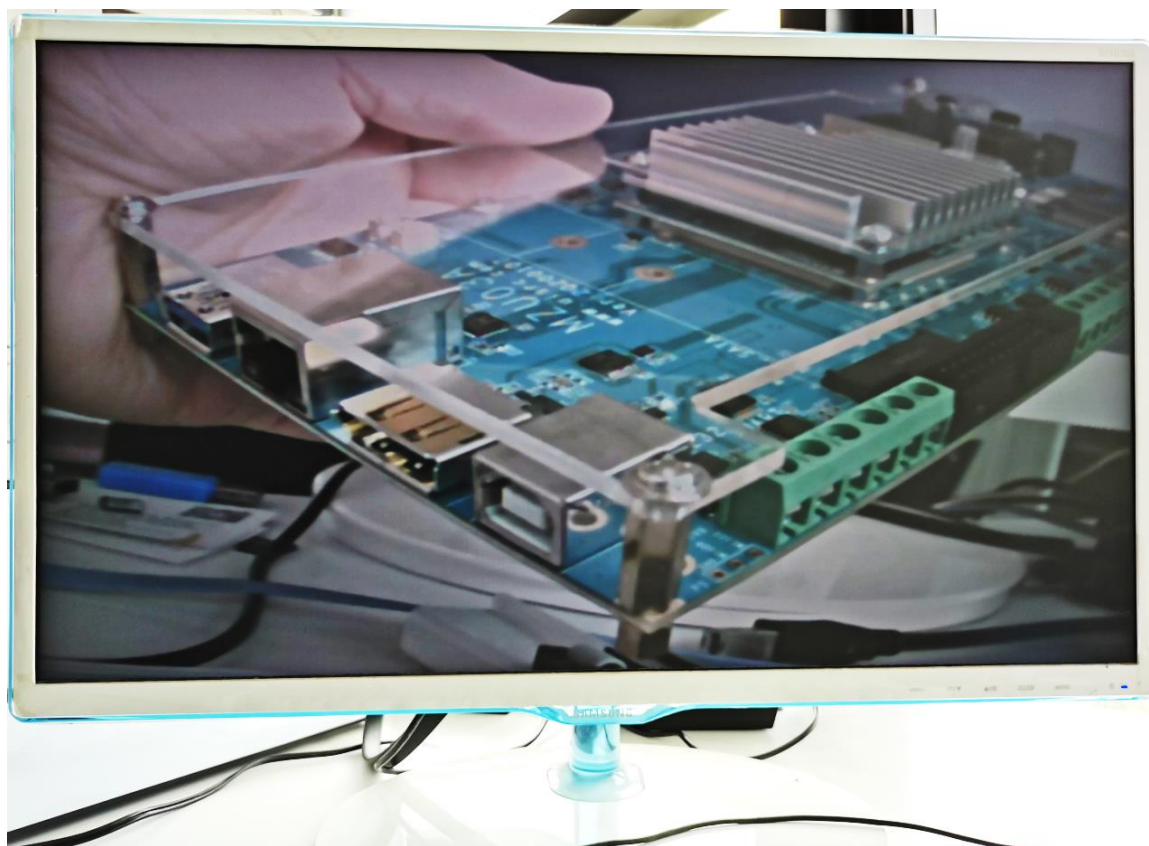
### 3.4 安装摄像头

扩展卡 FEP-Base-3.3V-Card 摄像头接法

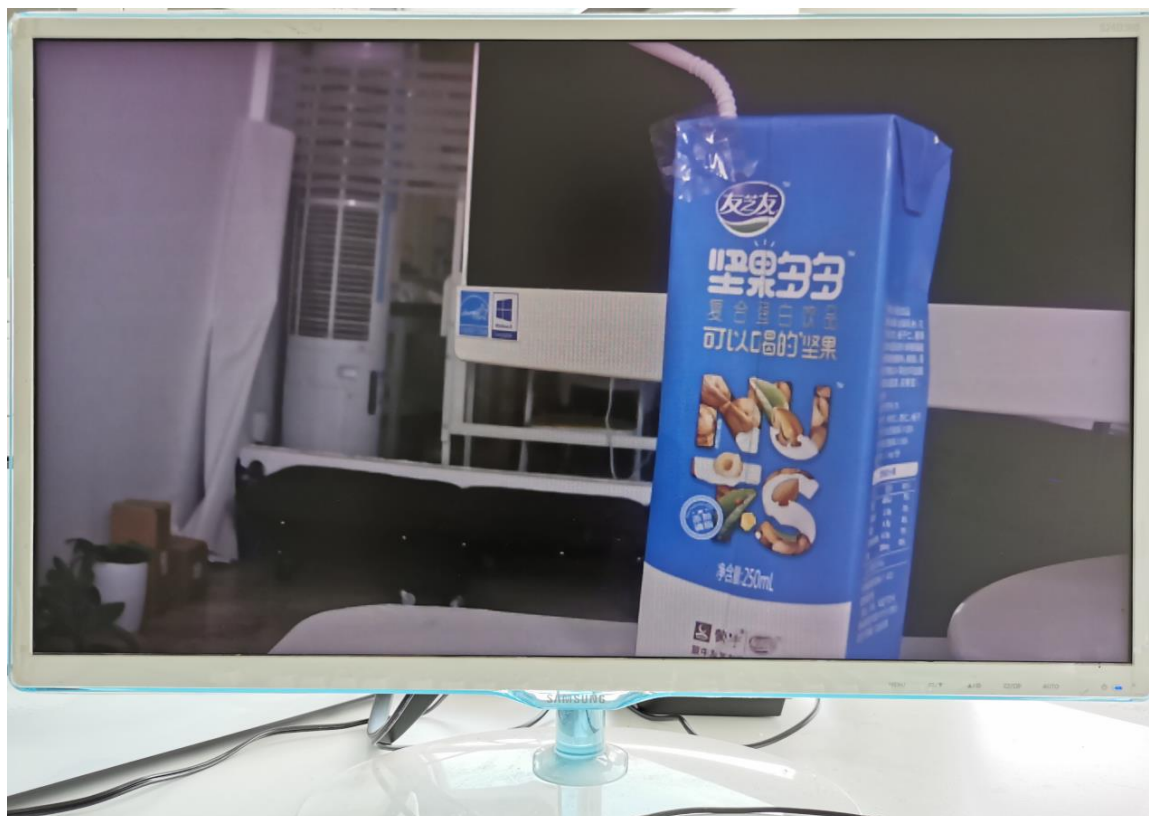


## 3.5 测试结果

### 3.5.1 0V7725



### 3.5.2 0V5640





### 3.5.3MT9V034





## 04HDMI 视频输入 DDR 三缓存

软件版本: VIVADO2019.2

操作系统: WIN10 64bit

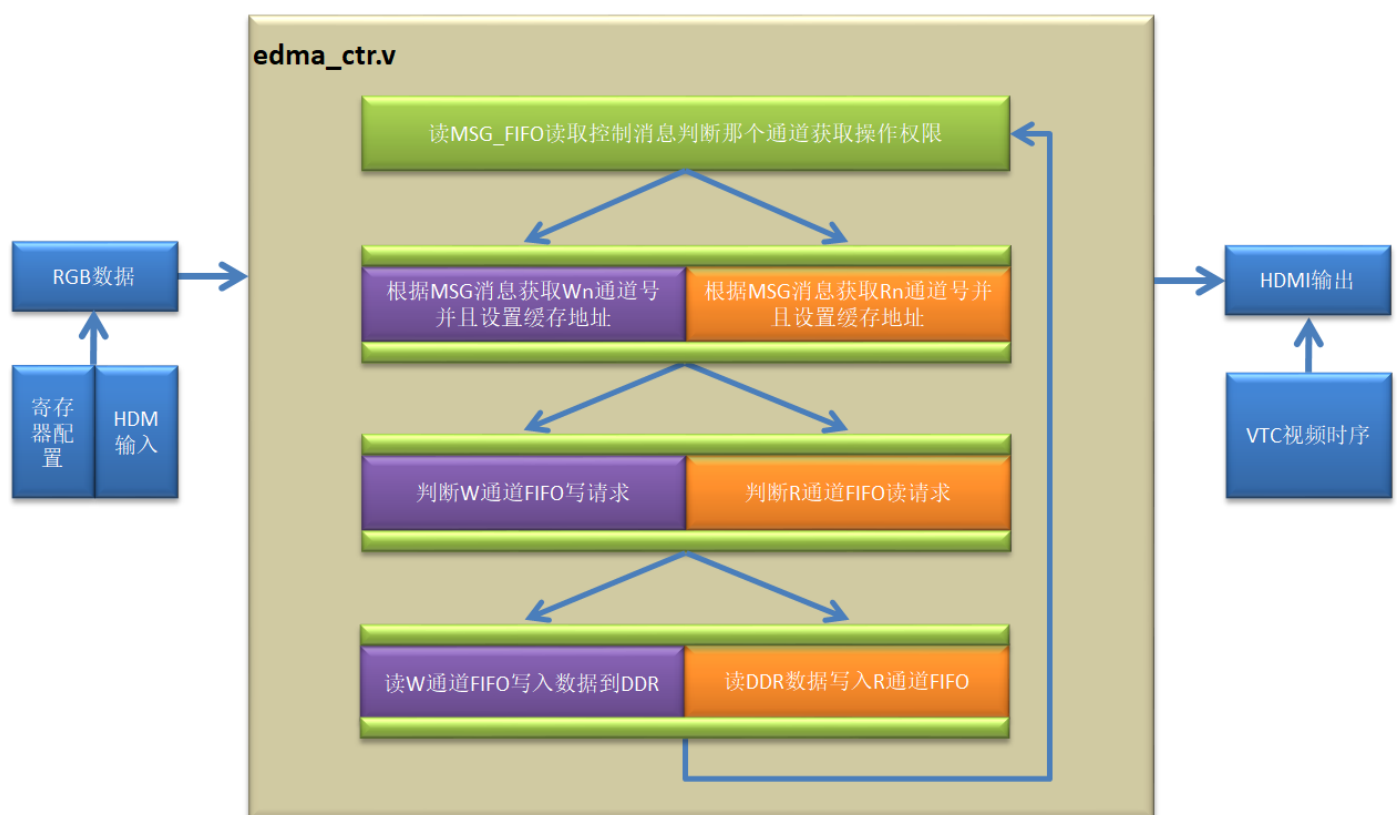
硬件平台: 适用 XILINX A7/K7/Z7/ZU/KU 系列 FPGA

登录米联客(MSXBO)FPGA 社区-[www.uisrc.com](http://www.uisrc.com) 观看免费视频课程、在线答疑解惑!

### 4.1 概述

基于前文的构架, 本文只要将前文代码稍加修改就能实现摄像头数据的缓存, 和摄像采集方案一样, HDMI 输入部分只是替换了数据输入的形式, 而本质一样。

### 4.2 构架设计



对于原来640\*480的输入分辨率, 本文中HDMI视频输入采用1920\*1080分辨率, 核心控制模块`edma_ctr.v`可以不做任何修改, 也可以增加每次burst的长度, 来提高效率。

1920\*1080分辨率参三分之一行数据:

parameter PKG\_SIZE =8'd80;//三分一行

parameter RD\_BURST\_LEN =8'd80;//一次 burst 80x256/32=640 个像素画

parameter WR\_BURST\_LEN =8'd80;//一次burst 80x256/32=640 个像素画

以上参数设置一行数据需要分3次burst

1920\*1080分辨率参数设置半行数据:

parameter PKG\_SIZE =8'd120;//半行像素

parameter RD\_BURST\_LEN =8'd120;//一次 burst 120x256/32= 960 个像素画

parameter WR\_BURST\_LEN =8'd120;//一次burst 120x256/32= 960个像素画

当然也可以设置每次burst一行数据, 并且需要增加w\_fifo和r\_fifo的深度。

## 4.3 代码结构

```
▼ ● img_hdmi_in (img_hdmi_in.v) (8)
  > > clk_wiz0_inst: clk_wiz_0 (clk_wiz_0.xci)
  > > clk_wiz1_inst: clk_wiz_1 (clk_wiz_1.xci)
  > > u_mig_7series_0: mig_7series_0 (mig_7series_0.xci)
  > ● uihdmix_inst: uihdmix (uihdmix.v) (8)
    ● uivtc_inst: uivtc (uivtc.v)
    ● uidelay: uidelay (uidelay.v)
  ▼ ● uicfg7611_inst: uicfg7611 (uicfg7611.v) (2)
    ● uii2c_inst: uii2c (uii2c.v)
    ● ui7611reg_inst: ui7611reg (ui7611reg.v)
  ▼ ● edma_ctr_inst: edma_ctr (edma_ctr.v) (5)
    ● fs_cap_W0: fs_cap (fs_cap.v)
    ● fs_cap_R0: fs_cap (fs_cap.v)
  > > ms_fifo_inst: ms_fifo (ms_fifo.xci)
  > > wr_fifo_inst: wr_fifo (wr_fifo.xci)
  > > rd_fifo_inst: rd_fifo (rd_fifo.xci)
```

我们主要关注下红色线框内的代码

uicfg7611.v ui7611reg.v uii2c.v 负责对HDMI输入芯片, ADV7611寄存器初始化, 设置输出分辨率是1920\*1080, 输出的视频格式为RGB888

## 4.4 HDMI 接线



- 1、开发板 HDMI IN 接口连接输入源: 测试使用 PC 机做输入源, HDMI 线连接开发板 HDMI IN 接口。
  - 2、开发板 HDMI OUT 接口连接输出源: 测试使用显示器做输出源, HDMI 线连接开发板 HDMI OUT 接口。
- 说明: 测试图片中的开发板不一定是本课程配套的开发板, 但是具有样例代表性。

## 4.5 测试结果



说明：测试图片中的开发板不一定是本课程配套的开发板，但是具有样例代表性。