DDR4 读写测试实验

黑金动力社区 2020-07-23

1 实验简介

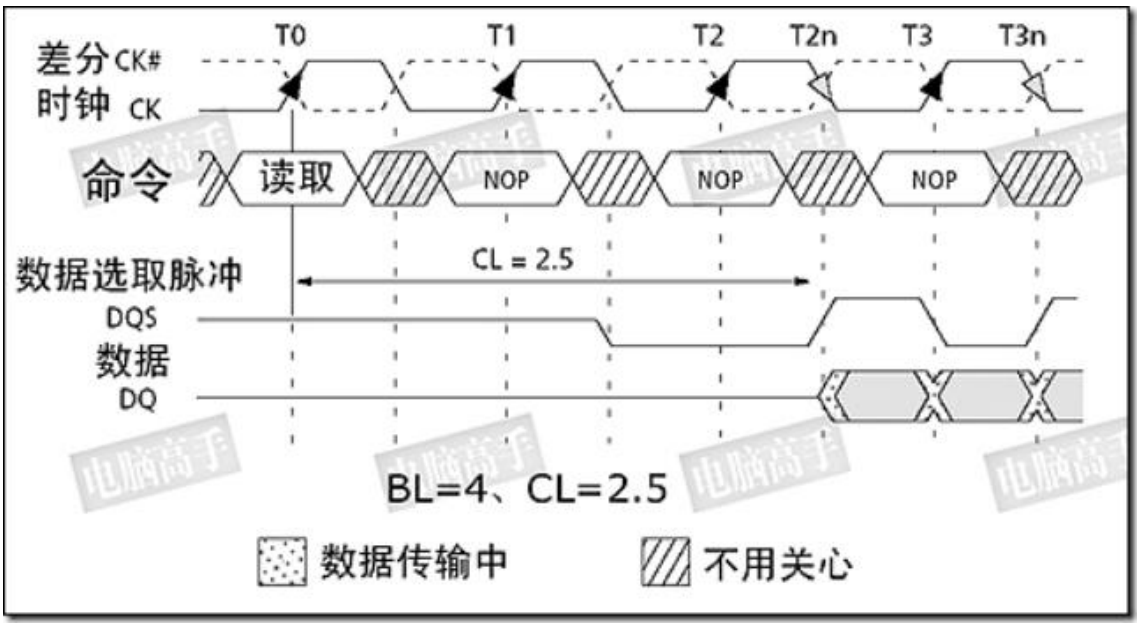
本实验为后续使用 DDR4 内存的实验做铺垫，通过循环读写 DDR4 内存，了解其工作原理和 DDR4 控制器的写法，由于 DDR4 控制复杂，控制器的编写难度高，这里笔者介绍 XILINX 的 MIG 控制器情况下应用，是后续音频、视频等需要用到 SDRAM 实验的基础。

2 实验原理

DDR SDRAM 全称为 Double Data Rate SDRAM，中文名为“双倍数据流 SDRAM”。DDR SDRAM 在原有的 SDRAM 的基础上改进而来。也正因为如此，DDR 能够凭借着转产成本优势来打败昔日的对手 RDRAM，成为当今的主流。本文只着重讲 DDR 的原理和 DDR SDRAM 相对于传统 SDRAM（又称 SDR SDRAM）的不同。

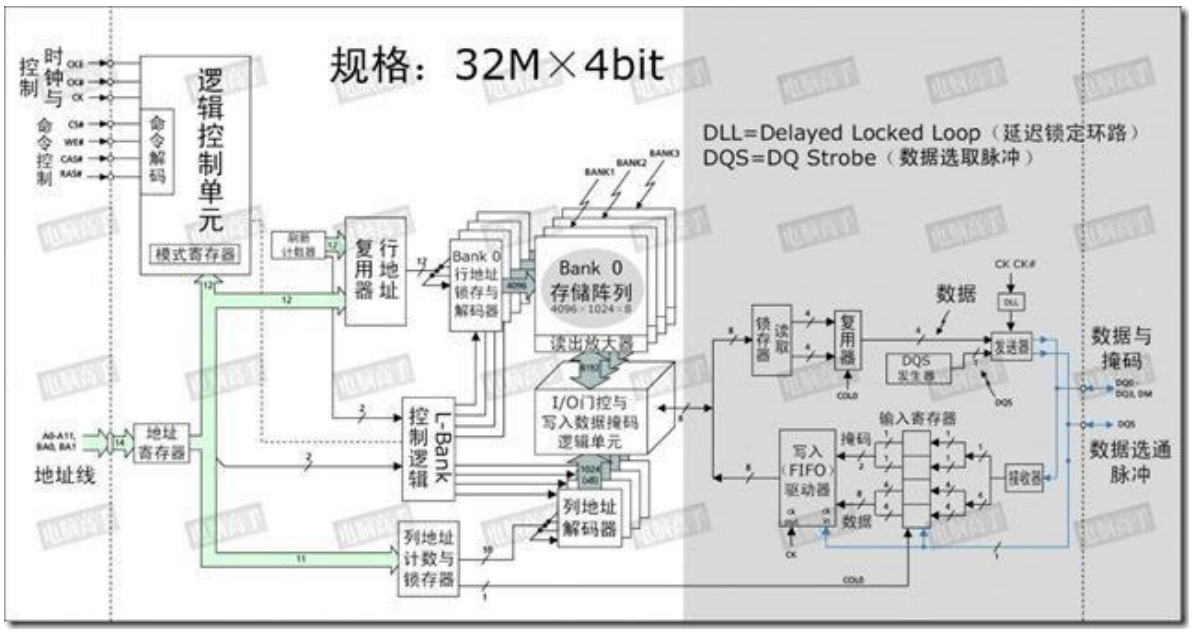
（一） DDR 的基本原理

有很多文章都在探讨 DDR 的原理，但似乎也不得要领，甚至还带出一些错误的观点。首先我们看看一张 DDR 正规的时序图。



从中可以发现它多了两个信号： CLK#与 DQS，CLK#与正常 CLK 时钟相位相反，形成差分时钟信号。而数据的传输在 CLK 与 CLK#的交叉点进行，可见在 CLK 的上升与下降沿（此时正好是 CLK# 的上升沿）都有数据被触发，从而实现 DDR。在此，我们可以说通过差分信号达到了 DDR 的目的，

甚至讲 CLK#帮助了第二个数据的触发，但这只是对表面现象的简单描述，从严格的定义上讲并不能这么说。之所以能实现 DDR，还要从其内部的改进说起。



DDR 内存芯片的内部结构图

这是一颗 128Mbit 的内存芯片，从图中可以看出来，白色区域内与 SDRAM 的结构基本相同，但请注意灰色区域，这是与 SDRAM 的不同之处。首先就是内部的 L-Bank 规格。SDRAM 中 L-Bank 存储单元的容量与芯片位宽相同，但在 DDR SDRAM 中并不是这样，存储单元的容量是芯片位宽的一倍，所以在此不能再套用讲解 SDRAM 时 “芯片位宽=存储单元容量” 的公式了。也因此，真正的行、列地址数量也与同规格 SDRAM 不一样了。

以本芯片为例，在读取时，L-Bank 在内部时钟信号的触发下一次传送 8bit 的数据给读取锁存器，再分成两路 4bit 数据传给复用器，由后者将它们合并为一路 4bit 数据流，然后由发送器在 DQS 的控制下在外部时钟上升与下降沿分两次传输 4bit 的数据给北桥。这样，如果时钟频率为 100MHz，那么在 I/O 端口处，由于是上下沿触发，那么就是传输频率就是 200MHz。

现在大家基本明白 DDR SDRAM 的工作原理了吧，这种内部存储单元容量（也可以称为芯片内部总线位宽）=2×芯片位宽（也可称为芯片 I/O 总线位宽）的设计，就是所谓的两位预取（2-bit Prefetch），有的公司则贴切的称之为 2-n Prefetch（n 代表芯片位宽）。

（二） DDR SDRAM 与 SDRAM 的不同

DDR SDRAM 与 SDRAM 的不同主要体现在以下几个方面。

DDR SDRAM 与 SDRAM 的主要不同对比表

内存类型 / 比较项	SDRAM	DDR SDRAM
命令		
全页式突发传输	支持	不支持
时钟信号挂起	支持	不支持
读出数据屏蔽	支持	不支持
写入数据屏蔽	支持	支持
功能		
时钟	单一时钟	差分时钟
预取设计	1-bit	2-bit
数据传输率	1/时钟周期	2/时钟周期
CAS 潜伏期	2、3	1.5、2、2.5、3
写入潜伏期	0	可变
突发长度	1、2、4、8、全页	2、4、8
延迟锁定回路	可选	工作时必需
自动刷新间隔周期	固定	弹性设计（最大值与 SDRAM 的固定值相同）
数据选取脉冲	无	必需
封装与电气特性		
封装类型（≥64Mbit）	TSOP-II 54pin（400mil）	TSOP-II 66pin（400mil） CSP 60pin
工作电压	3.3V（LVTTL 接口）	2.5V（SSTL_2 接口）
模组	168pin DIMM	184pin DIMM

注：LVTTL=Low Voltage Transistor-Transistor Logic（低电压晶体管-晶体管逻辑电路）、SSTL=Stub Series Terminated Logic（短线串联终止逻辑电路）

提示：TSOP-II 与 CSP
TSOP-II：是指小外形薄型封装（Thin Small Outline Package）的第二种方式，引脚在封装的长边两侧，TSOP-I 的引脚则在短边的两侧。 CSP：是指芯片尺寸封装（Chip Scale Package），其封装尺寸与芯片核心尺寸基本相同，所以称 CSP，其内核面积与封装面积的比例约为 1:1.1，凡是符合这一标准的封装都可称之为 CSP。

DDR SDRAM 与 SDRAM 一样，在开机时也要进行 MRS，不过由于操作功能的增多，DDR SDRAM 在 MRS 之前还多了一 EMRS 阶段（Extended Mode Register Set，扩展模式寄存器设置），这个扩展模式寄存器控制着 DLL 的有效/禁止、输出驱动强度、QFC 有效/无效等。

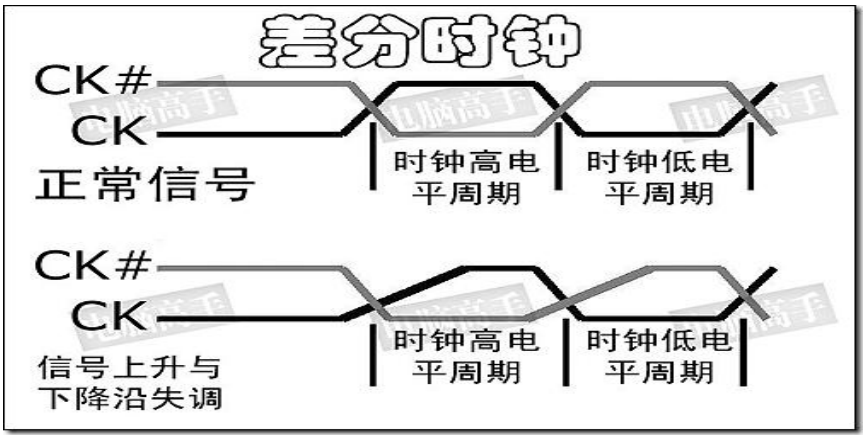
提示与误区：QFC 的含义与作用
QFC 是指 FET Switch Controller（FET 开关控制），低电平有效。用于借助外部 FET（场效应管）开关控制模组上芯片间的相互隔离，没有读写操作时进入隔离状态，以确保芯片间不受相互干扰。QFC 是一个特选功能，厂商都是在接到芯片买家的指定要求后，才在芯片中加入这一功能，并且需要在模装配时进行相关的设计改动（如增加 VddQ 的上拉电阻），所以 DIY 市场上几乎很少见到支持这一功能的 DDR 内存。而在 2002 年 5 月，JEDEC 最新发布的 DDR 规范中，已经不在有 QFC 的定义，而且即使有 FET 开关也将由北桥控制（此时是用来隔离模组的），因此 QFC 已经成为历史。可是，在很多“深入”性介绍中，都将 QFC 认为是一个必要的过程，这显然是错的。

由于 EMRS 与 MRS 的操作方法与 SDRAM 的 MRS 大同小异，在此就不再列出具体的模式表了，有兴趣的话可查看相关的 DDR 内存资料。下面我们就着重说说 DDR SDRAM 的新设计与新功能。

误区：CSP 与 uBGA、WBGA、TinyBGA、FBGA 等是不同的封装技术
实际上，CSP 只是一种封装标准/类型，不涉及具体的封装技术，只要达到它的尺寸标准都可称之为 CSP 封装。近几年出现的 uBGA、WBGA、TinyBGA、FBGA 小型芯片封装技术则是 CSP 的具体表现形式（其实都是 BGA 封装技术的一种），由此可以看出 CSP 并没有固定的封装技术，它自己更不是一个封装技术，只要厂商愿意或有实力，可以开发出更多的符合 CSP 标准的封装技术。

1、 差分时钟

差分时钟（参见上文“DDR SDRAM 读操作时序图”）是 DDR 的一个必要设计，但 CK#的作用，并不能理解为第二个触发时钟（你可以在讲述 DDR 原理时简单地这么比喻），而是起到触发时钟校准的作用。由于数据是在 CK 的上下沿触发，造成传输周期缩短了一半，因此必须要保证传输周期的稳定以确保数据的正确传输，这就要求 CK 的上下沿间距要有精确的控制。但因为温度、电阻性能的改变等原因，CK 上下沿间距可能发生变化，此时与其反相的 CK#就起到纠正的作用（CK 上升快下降慢，CK# 则是上升慢下降快）。而由于上下沿触发的原因，也使 CL=1.5 和 2.5 成为可能，并容易实现。与 CK 反相的 CK#保证了触发时机的准确性。



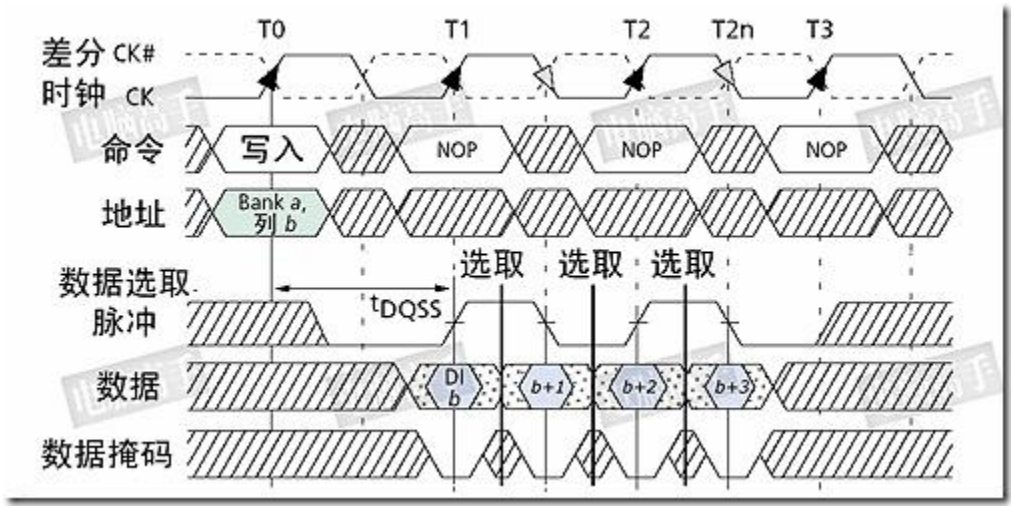
2、 数据选取脉冲（DQS）

总结 DQS：它是双向信号；读内存时，由内存产生，DQS 的沿和数据的沿对齐；写入内存时，由外部产生，DQS 的中间对应数据的沿，即此时 DQS 的沿对应数据最稳定的中间时刻。

DQS 是 DDR SDRAM 中的重要功能，它的功能主要用来在一个时钟周期内准确的区分出每个传输周期，并便于接收方准确接收数据。每一颗芯片都有一个 DQS 信号线，它是双向的，在写入时它用来传送由北桥发来的 DQS 信号，读取时，则由芯片生成 DQS 向北桥发送。完全可以说，它就是数据的同步信号。

在读取时，DQS 与数据信号同时生成（也是在 CK 与 CK#的交叉点）。而 DDR 内存中的 CL 也就是从 CAS 发出到 DQS 生成的间隔，数据真正出现在数据 I/O 总线上相对于 DQS 触发的时间间隔被称为 tAC。注意，这与 SDRAM 中的 tAC 的不同。实际上，DQS 生成时，芯片内部的预取已经完毕了，tAC 是指上文结构图中灰色部分的数据输出时间，由于预取的原因，实际的数据传出可能会提前于 DQS 发生（数据提前于 DQS 传出）。由于是并行传输，DDR 内存对 tAC 也有一定的要求， 对于 DDR2，tAC 的允许范围是±0.75ns，对于 DDR3，则是±0.7ns，有关它们的时序图示见前文，其中 CL 里包含了一段 DQS 的导入期。

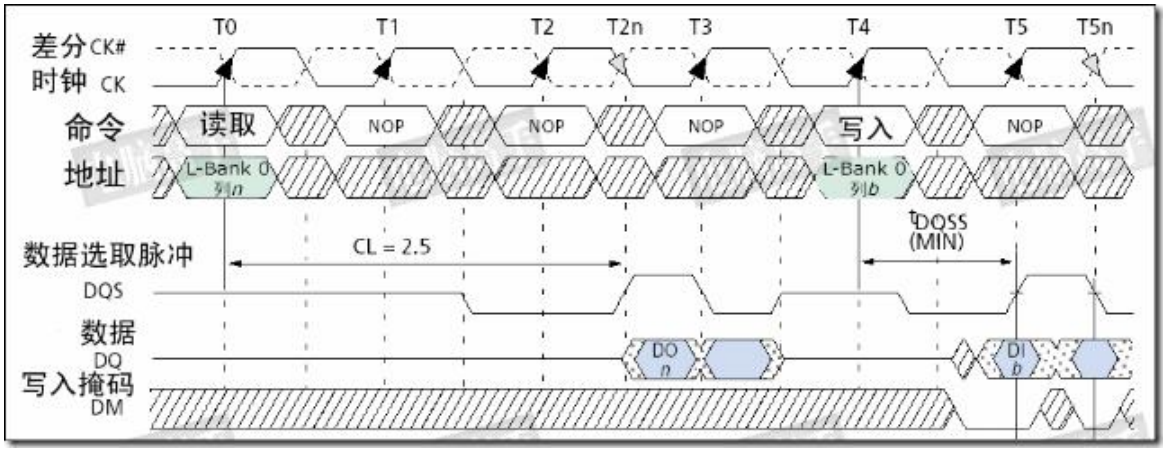
前文已经说了 DQS 是为了保证接收方的选择数据， DQS 在读取时与数据同步传输，那么接收时也是以 DQS 的上下沿为准吗？不，如果以 DQS 的上下沿区分数据周期的危险很大。由于芯片有 预取的操作，所以输出时的同步很难控制，只能限制在一定的时间范围内，数据在各 I/O 端口的出现时间可能有快有慢，会与 DQS 有一定的间隔，这也就是为什么要有一个 tAC 规定的原因。而在接收方，一切必须保证同步接收，不能有 tAC 之类的偏差。这样在写入时，芯片不再自己生成DQS，而以发送方传来的 DQS 为基准，并相应延后一定的时间，在 DQS 的中部为数据周期的选取分割点（在读取时分割点就是上下沿），从这里分隔开两个传输周期。这样做的好处是，由于各 数据信号都会有一个逻辑电平保持周期，即使发送时不同步，在 DQS 上下沿时都处于保持周期中， 此时数据接收触发的准确性无疑是最高的。在写入时，以 DQS 的高/低电平期中部为数据周期分割点，而不是上/下沿，但数据的接收触发仍为 DQS 的上/下沿。



3、 写入延迟

在上面的 DQS 写入时序图中，可以发现写入延迟已经不是 0 了，在发出写入命令后，DQS 与写入数据要等一段时间才会送达。这个周期被称为 DQS 相对于写入命令的延迟时间（tDQSS，WRITE Command to the first corresponding rising edge of DQS），对于这个时间大家应该很好理解了。

为什么要有这样的延迟设计呢？原因也在于同步，毕竟一个时钟周期两次传送，需要很高的 控制精度，它必须要等接收方做好充分的准备才行。tDQSS 是 DDR 内存写入操作的一个重要参数，太短的话恐怕接受有误，太长则会造成总线空闲。tDQSS 最短不能小于 0.75 个时钟周期，最长不能超过 1.25 个时钟周期。有人可能会说，如果这样，DQS 不就与芯片内的时钟不同步了吗？对， 正常情况下，tDQSS 是一个时钟周期，但写入时接受方的时钟只用来控制命令信号的同步，而数 据的接受则完全依靠 DQS 进行同步，所以 DQS 与时钟不同步也无所谓。不过，tDQSS 产生了一个不利影响——读后写操作延迟的增加，如果 CL=2.5，还要在 tDQSS 基础上加入半个时钟周期，因为命令都要在 CK 的上升沿发出。



当 CL=2.5 时，读后写的延迟将为 tDQSS+0.5 个时钟周期（图中 BL=2）

另外，DDR 内存的数据真正写入由于要经过更多步骤的处理，所以写回时间（tWR）也明显延长，一般在 3 个时钟周期左右，而在 DDR-Ⅱ规范中更是将 tWR 列为模式寄存器的一项，可见它的重要性。

误区：DDR SDRAM 各种延迟与潜伏期的单位时间减半
一些文章认为，DDR 使数据传输率加倍，那么与之相关的延迟与潜伏期的单位时间也减半，比如 DDR-266 内存，tRCD、CL、tRP 的单位周期为 3.75ns，比 PC133 内存少了一半。这是严重的概念性错误，从我们列举的时序图中可以看出，tRCD、CL、tRP 是以时钟信号来界定的，不能用传输周期去表示，否则 CL=2.5 的参考基准是什么？对于 DDR-266，时钟频率是 133MHz，时钟周期仍是 7.5，和 PC133 的标准一样。那些文章的作者显然是将时钟周期与传输周期弄混了

4、 突发长度与写入掩码

在 DDR SDRAM 中，突发长度只有 2、4、8 三种选择，没有了随机存取的操作（突发长度为 1）和全页式突发。这是为什么呢？因为 L-Bank 一次就存取两倍于芯片位宽的数据，所以芯片至少也

要进行两次传输才可以，否则内部多出来的数据怎么处理？而全页式突发事实证明在 PC 内存中是很难用得上的，所以被取消也不希奇。

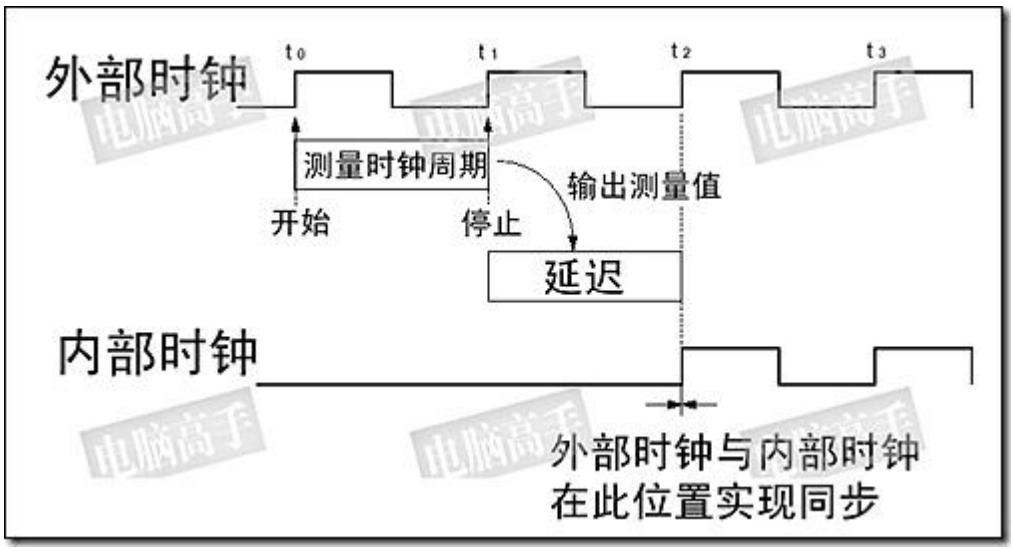
但是，突发长度的定义也与 SDRAM 的不一样了（见本章节最前那幅 DDR 简示图），它不再指所连续寻址的存储单元数量，而是指连续的传输周期数，每次是一个芯片位宽的数据。对于突发写入，如果其中有不想存入的数据，仍可以运用 DM 信号进行屏蔽。DM 信号和数据信号同时发出，接收方在 DQS 的上升与下降沿来判断 DM 的状态，如果 DM 为高电平，那么之前从 DQS 中部选取的数据就被屏蔽了。有人可能会觉得，DM 是输入信号，意味着芯片不能发出 DM 信号给北桥作为屏蔽读取数据的参考。其实，该读哪个数据也是由北桥芯片决定的，所以芯片也无需参与北桥的工作，哪个数据是有用的就留给北桥自己去选吧。

5、 延迟锁定回路（DLL）

DDR SDRAM 对时钟的精确性有着很高的要求，而 DDR SDRAM 有两个时钟，一个是外部的总线时钟，一个是内部的工作时钟，在理论上 DDR SDRAM 这两个时钟应该是同步的，但由于种种原因，如温度、电压波动而产生延迟使两者很难同步，更何况时钟频率本身也有不稳定的情况（SDRAM 也内部时钟，不过因为它的工作/传输频率较低，所以内外同步问题并不突出）。DDR SDRAM 的 tAC 就是因为内部时钟与外部时钟有偏差而引起的，它很可能造成因数据不同步而产生错误的恶果。实际上，不同步就是一种正/负延迟，如果延迟不可避免，那么若是设定一个延迟值，如一个时钟周期，那么内外时钟的上升与下降沿还是同步的。鉴于外部时钟周期也不会绝对统一，所以需要根据外部时钟动态修正内部时钟的延迟来实现与外部时钟的同步，这就是 DLL 的任务。

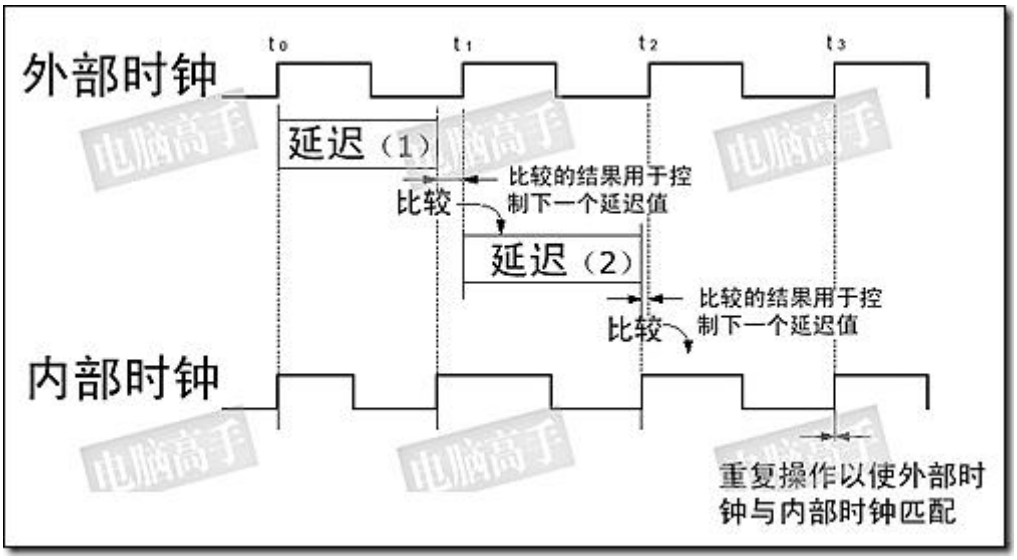
DLL 不同于主板上的 PLL，它不涉及频率与电压转换，而是生成一个延迟量给内部时钟。目前 DLL 有两种实现方法，一个是时钟频率测量法（CFM，Clock Frequency Measurement），一个是时钟比较法（CC，Clock Comparator）。

CFM 是测量外部时钟的频率周期，然后以此周期为延迟值控制内部时钟，这样内外时钟正好就相差了一个时钟周期，从而实现同步。DLL 就这样反复测量反复控制延迟值，使内部时钟与外部时钟保持同步。



CFM 式 DLL 工作示意图

CC 的方法则是比较内外部时钟的长短，如果内部时钟周期短了，就将所少的延迟加到下一个内部时钟周期里，然后再与外部时钟做比较，若是内部时钟周期长了，就将多出的延迟从下一个内部时钟中刨除，如此往复，最终使内外时钟同步。



CC 式 DLL 工作示意图

CFM 与 CC 各有优缺点，CFM 的校正速度快，仅用两个时钟周期，但容易受到噪音干扰，并且如果测量失误，则内部的延迟就永远错下去了。CC 的优点则是更稳定可靠，如果比较失败，延迟受影响的只是一个数据（而且不会太严重），不会涉及到后面的延迟修正，但它的修正时间要比 CFM 长。DLL 功能在 DDR SDRAM 中可以被禁止，但仅限于除错与评估操作，正常工作状态是自动有效的。

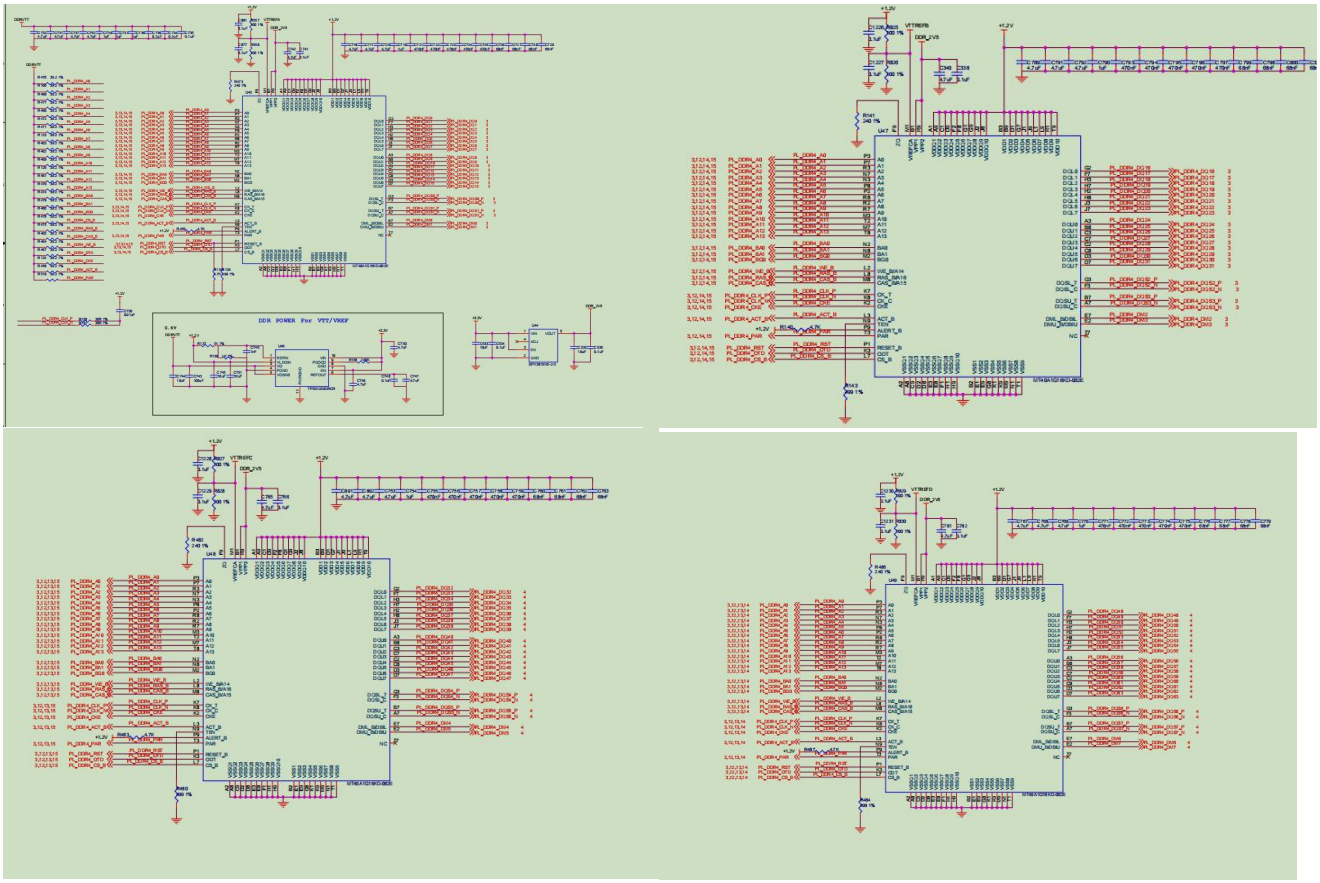
误区：DLL 是实现 DDR 传输的关键
“DDR 内存通过内部的 DLL 延时锁相环提供精确的时钟定位，这样就可以在每个时钟周期的上升沿和下降沿传输数据”。“DDR 使用了 DLL 来提供一个数据滤波信号 DQS 来选取数据”。 以上是目前较为流行的对 DDR 工作原理的两种解释，现在大家能看出错误所在吗？两者都把 DLL 的功能夸大了，DLL 只是一个重要的辅助校准设计，与能否实现双沿触发没有关系，它只是保证数据的输出尽量与外部时钟同步。后者则是概念性错误，从 DDR 内存结构图可看出，DQS 不是 DLL 生成的，只是由 DLL 保证其与外部时钟的同步，DQS 由单独的 DQS 发生器生成。

3 硬件介绍

AXKU040 开发板上使用了 4 个 Micron DDR4 的颗粒 MT40A1G16KD-062E,每个 DDR 芯片的容量为4Gb。4 个 DDR4 芯片组合成 64 位的数据总线宽度和 FPGA 相连接。开发板板上对 DDR4 的地址线和控制线都做了端接电阻上拉到 VTT 电压，保证信号的质量。在 PCB 的设计上，完全遵照 XILINX 的

DDR4 设计规范, 严格保证等长设计和阻抗控制。FPGA 的 DDR 管脚分配是要有所考虑的，而不能随意分配。如果用户自己实在不清楚怎么连接，那就请完全参考我们的原理图来设计，依样画葫芦总会的吧？

在 PCB 的设计上, 考虑高速信号的数据传输的可靠性, 走线上严格保证等长设计和阻抗控制。开发板 DDR 部分的原理图如下:



AXKU040 开发板 DDR4

4 程序设计

MIG 控制器设计

MIG IP 控制器是 Xilinx 为用户提供的一个 DDR 控制的 IP, 这样用户即使不了解 DDR 的控制和读写时序也能通过 DDR 控制器方便的读写 DDR 存储器。7 系列的 DDR 控制器的解决方案如下所示

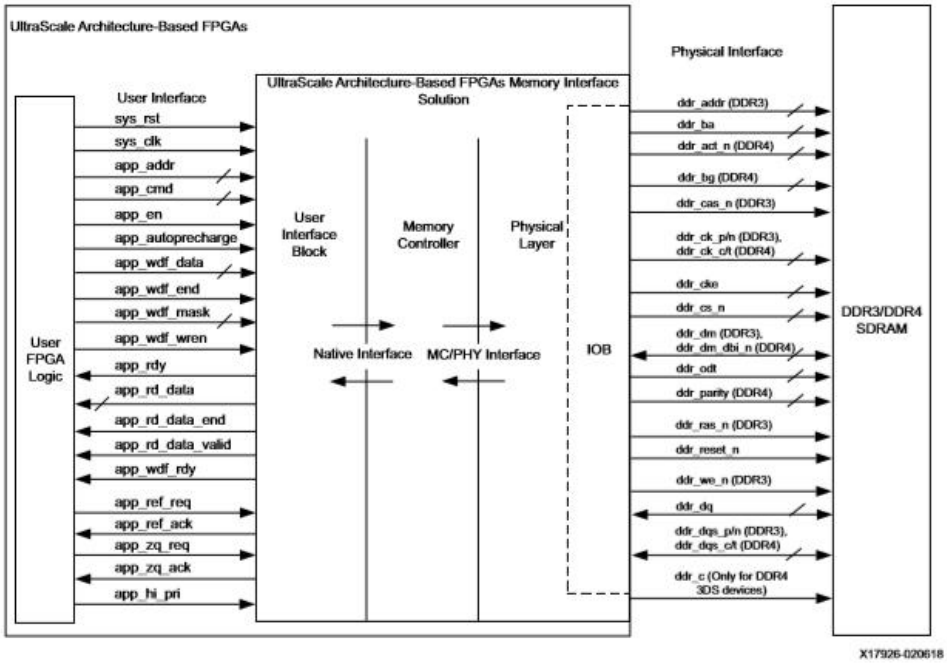
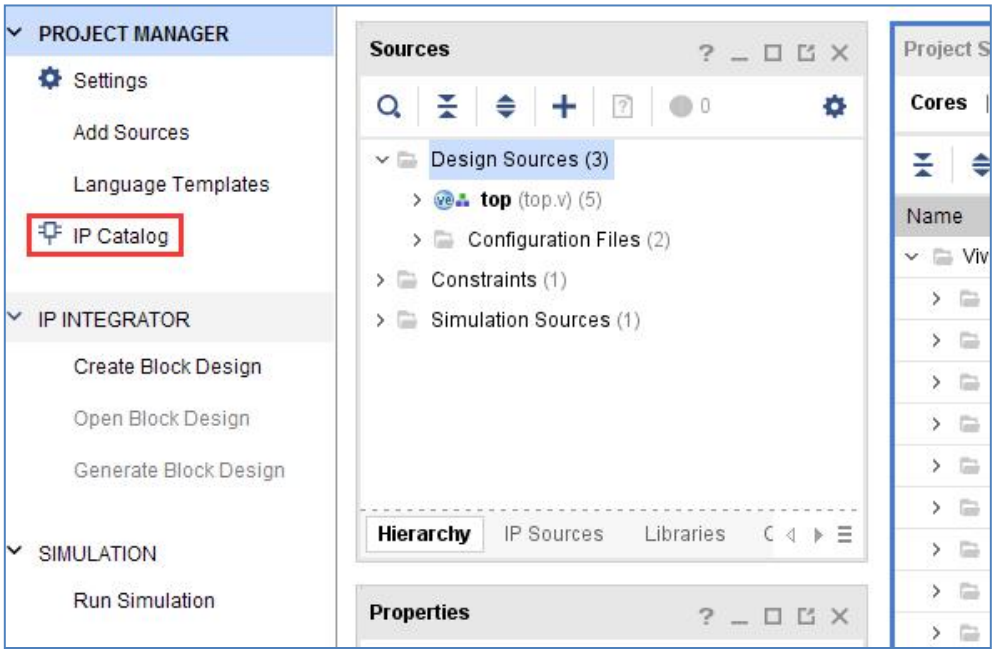


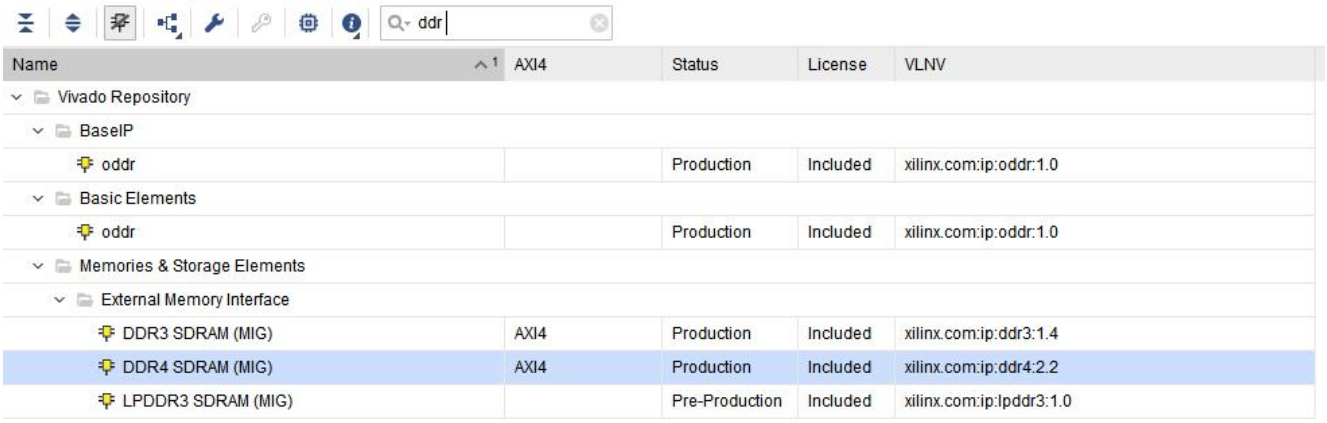
Figure 1-1: **UltraScale Architecture-Based FPGAs DDR3/DDR4 Memory Interface Solution**

DDR4控制器包含 3 部分:用户接口模块(User interface Block), 存储器控制模块(Memory Controller)和 DDR3 的物理接口(Physical Layer)。开发人员只需要开发用户的逻辑设计跟 DDR 控制器的用户接口对接来读写 DDR4 的数据。关于 DDR4 控制器用户端的接口定义和时序的更多介绍，大家还是参考 Xilinx 提供的文档（UG150），接下来为大家介绍如何生成和配置 DDR4 控制器吧！

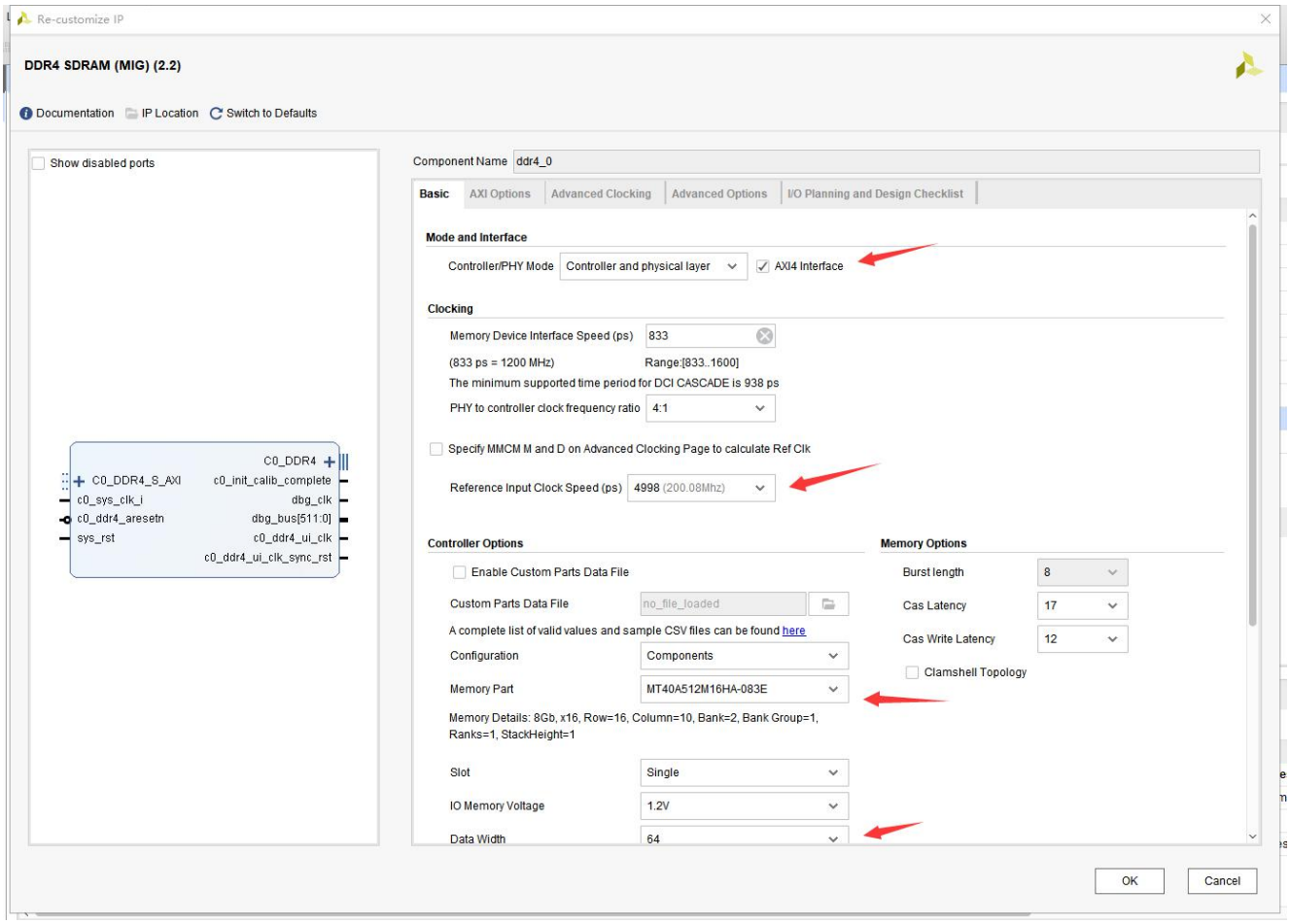
1. 首先在 Vivado 环境里新建一个项目，取名为 ddr4_test。再点击 Project Manager 界面下的 IP Catalog，打开 IP Catalog 界面。



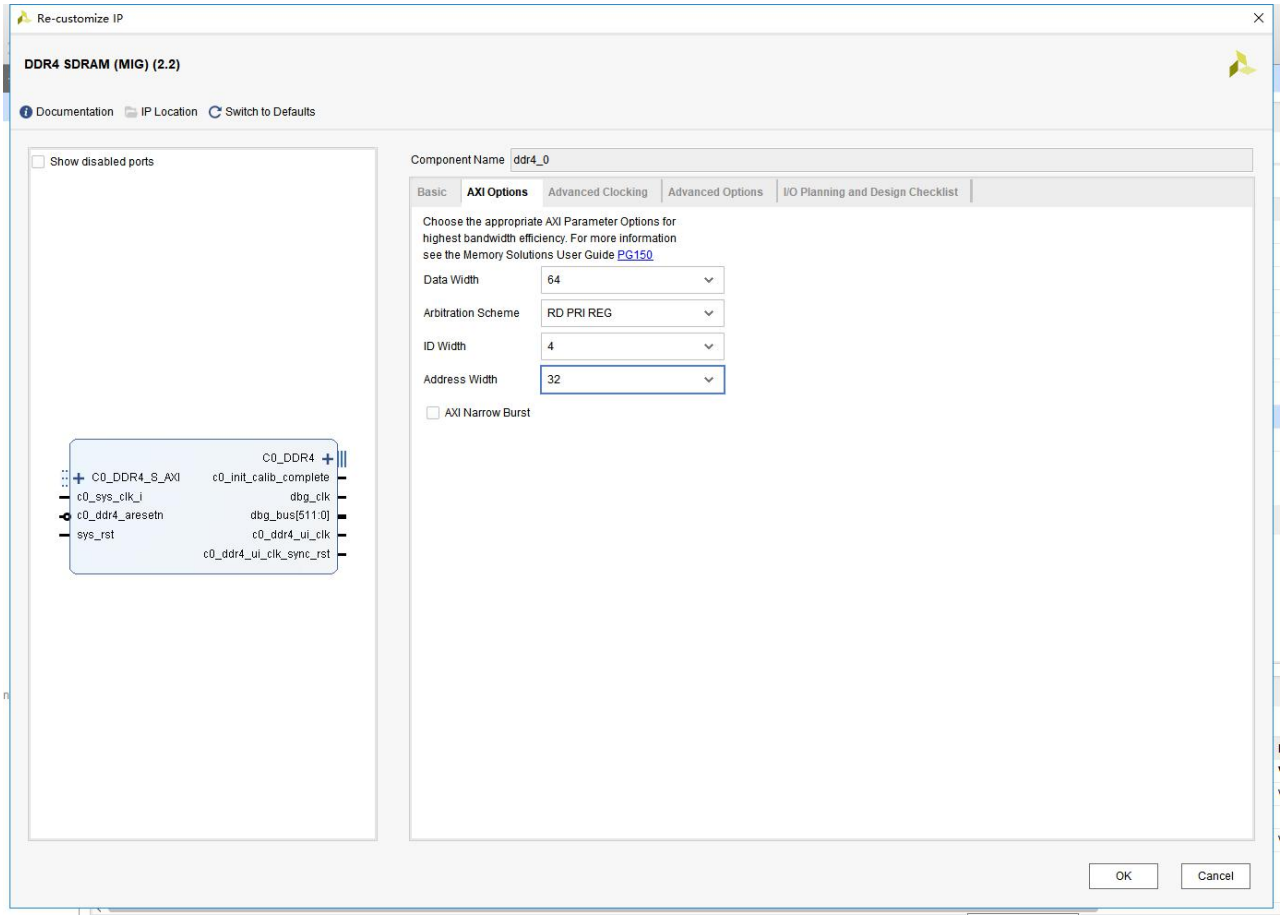
1.在 IP Catalog 界面里搜索ddr4并选择



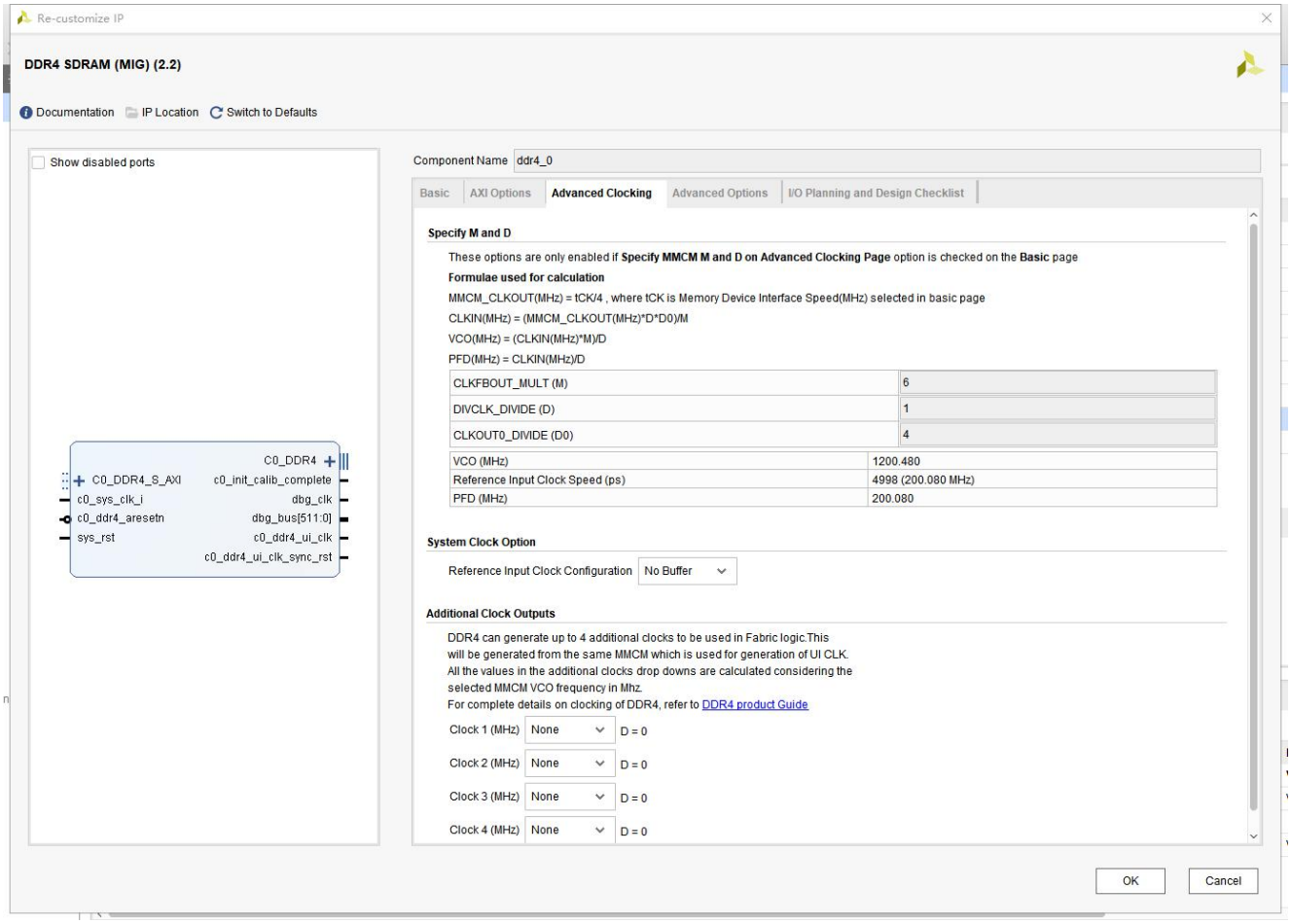
2.在basic栏里这 选择AXI4 Interface 参考时钟我们选择开发板对应的系统时钟 200M Memory由于版本原因这里选择相近DDR型号 MT40A512M16HA-083E Data Width选择64位



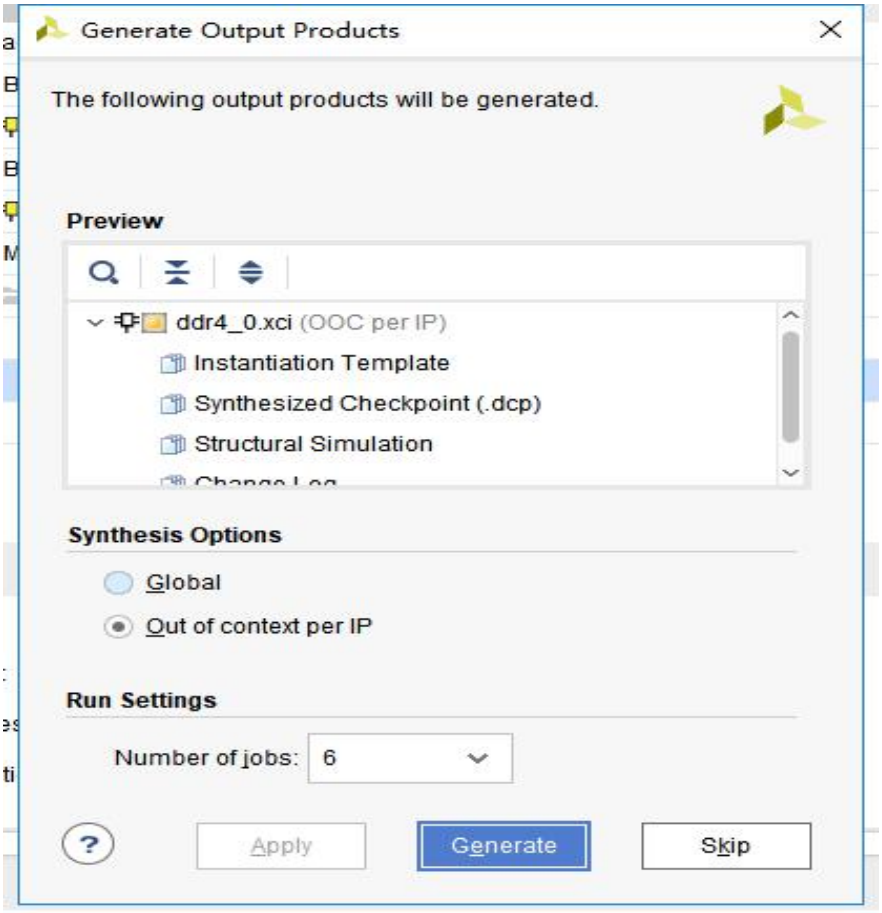
2. AXI Options里也需要进行对应的位宽，位宽的大小根据程序设计设置大家也可以参考官方提供的PG150文档进行设计



3. 在Advanced Clocking这里我们将系统时钟设置No Buffer



3. 其余俩项默认 “ok” 点击Generate 生成ip


Generate Output Products
The following output products will be generated.
Preview
ddr4_0.xci (OOC per IP)
Instantiation Template
Synthesized Checkpoint (.dcp)
Structural Simulation
Synthesis Options
Global
Out of context per IP
Run Settings
Number of jobs:
6
Apply
Generate
Skip

5 DDR4 测试程序设计

如果要让大家在自己编写一个 DDR4的测试程序，或者在自己的程序中使用这个 DDR4 IP 的话，相信很多同学还是会找不到北，不知从何下手了。那这节我们就来学习自己编写一个 DDR4 的驱动程序来跟 DDR4 IP 进行通信， 来实现DDR4数据读写。

1. 接下去我们来编写一个顶层程序 top.v，在顶层模块里面我们分别例化了mem_test、aq_axi_master、以及例化的ddr4ip

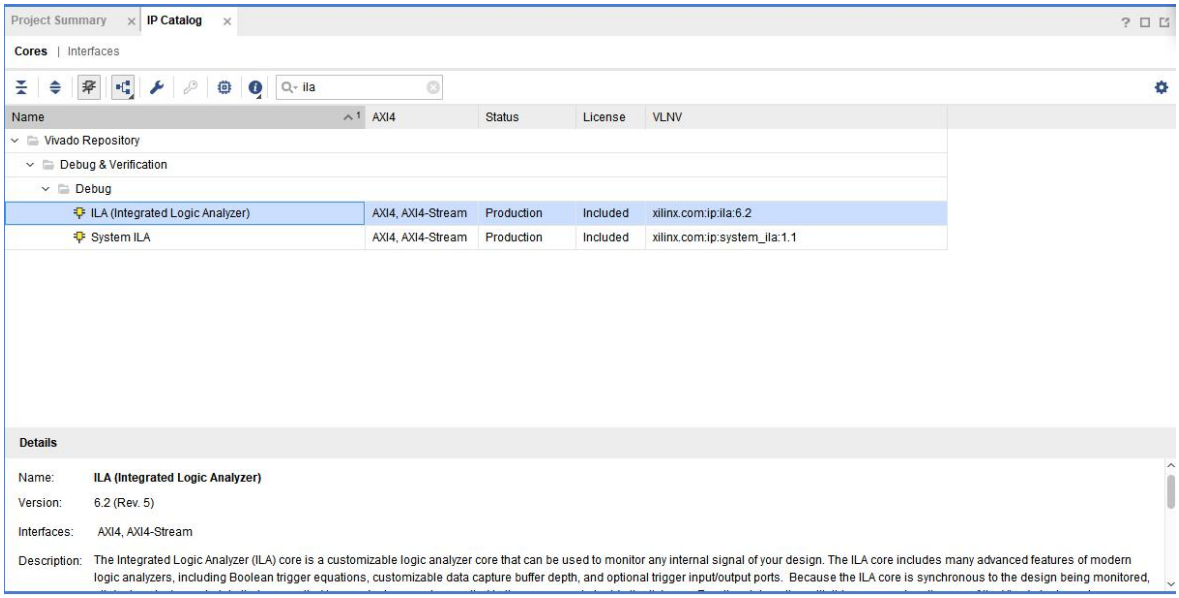
2.aq_axi_master主要与我们ddr4进行通信，可以看到与这个模块接口比较多代码没有多少行通过axi协议与ddr4进行通信。想了解这部分协议可以参考资料里提供的文档 AXI_specification里面对接口以及协议做了详细的描述

3. 接下去我们来编写一个 DDR4 的测试程序 mem_test.v

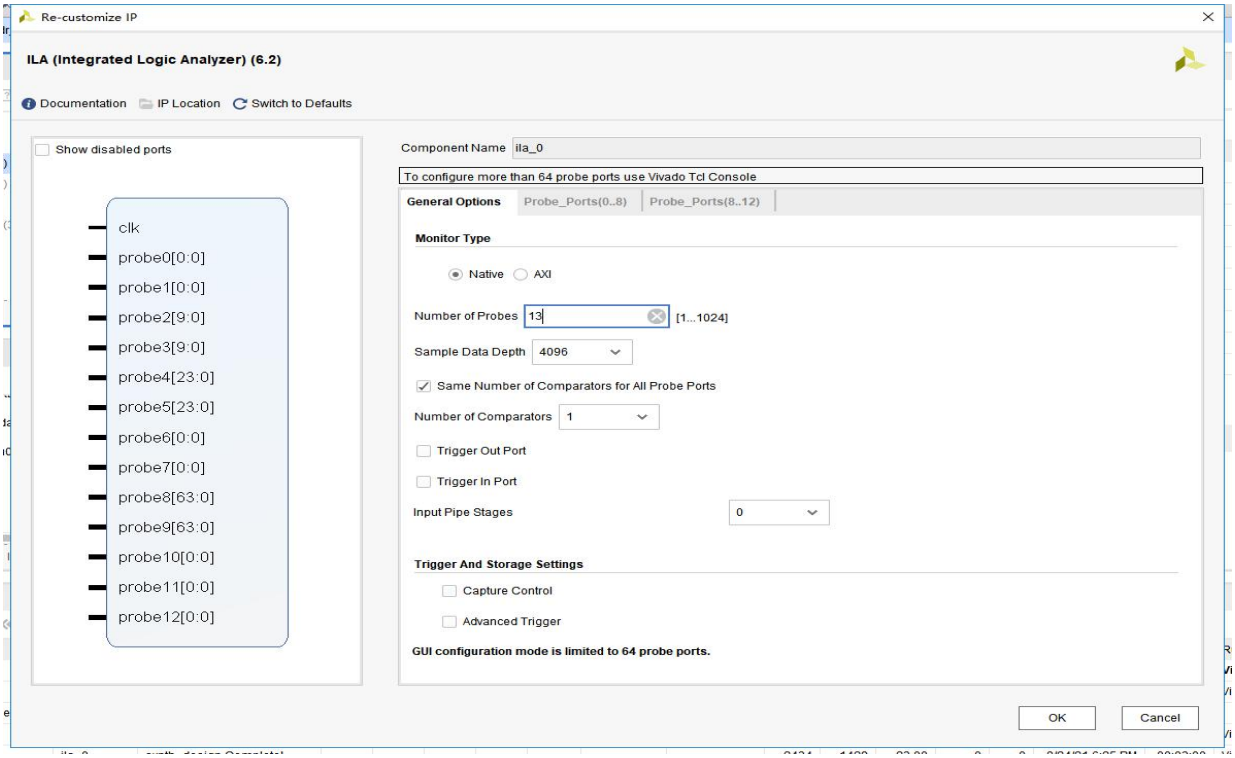
mem_test.v 测试程序里主要实现 ddr 的 burst 读和 burst 写的功能, 程序里产生读写请求信号, 地址和测试数据并校验读和写的数据是否正确, 这里 burst 的数据长度是 128。如果 DDR 的读写数据错误(写入的数据和读出的数据不一致），error 信号会变高。

具体的读写时序大家可以结合 ila 看具体波形，这样容易理解一些。

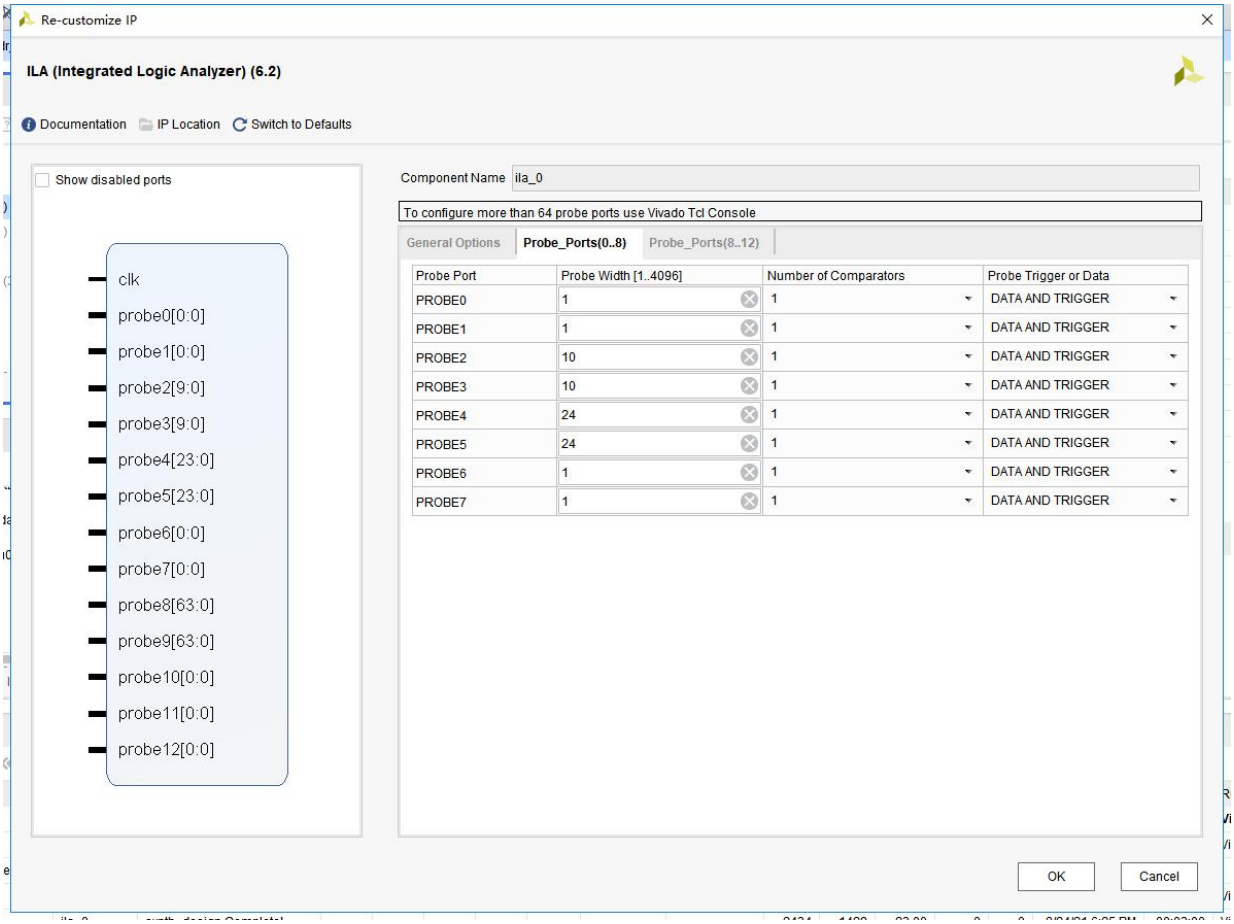
2. 添加 ila IP，打开 IP Catalog 界面，选择搜索下 ILA（Integrated Logic Analyzer）,双击打开。



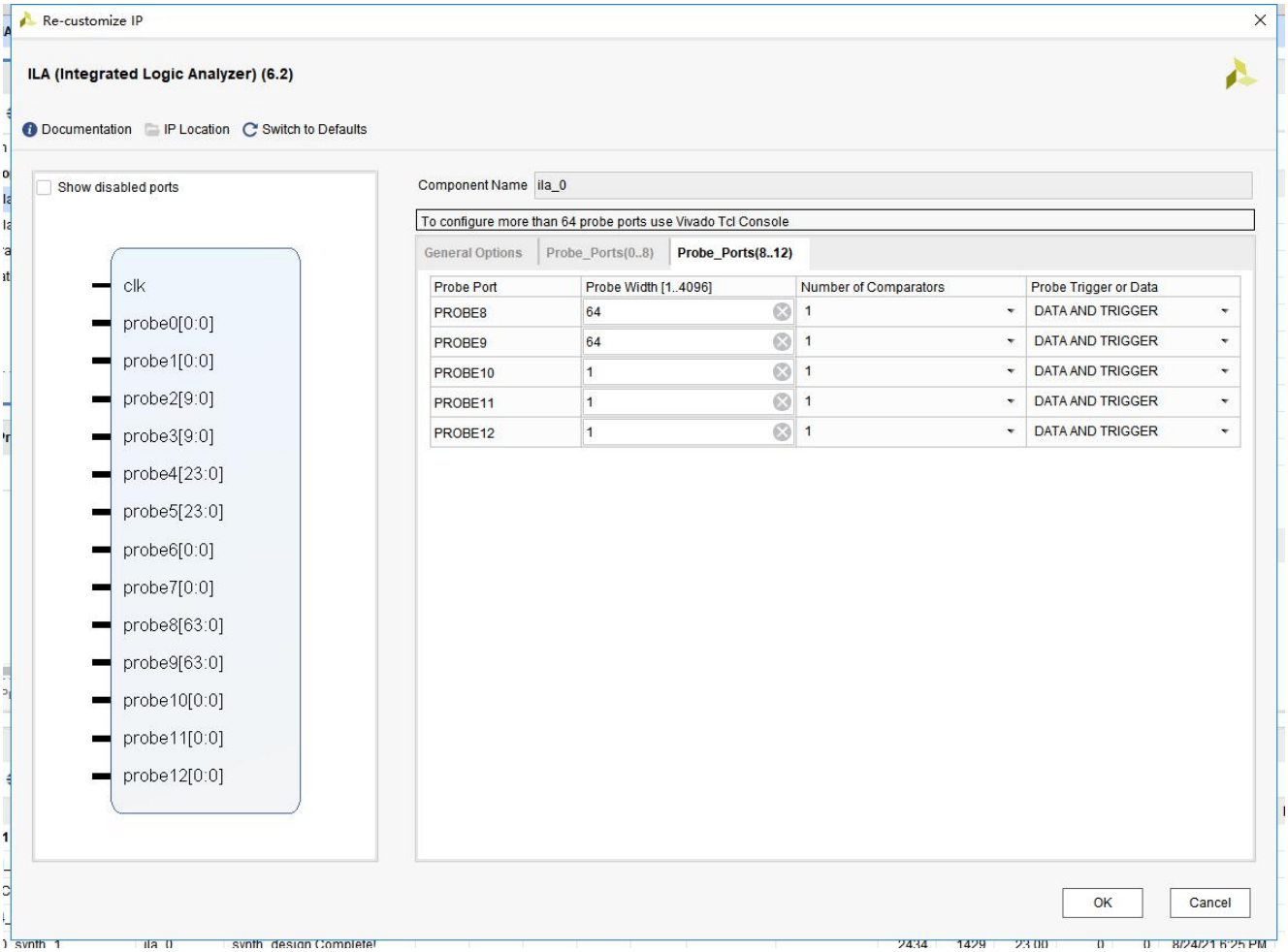
Component 名字为 ila_0，名字需要跟程序里的一样。Probe 的数量跟你想要采集的信号而定采样通道这里设置的13，采样的数据深度为 4096, 采样深度越深，采样的数据量越大，但会消耗更多的 FPGA 逻辑资源。



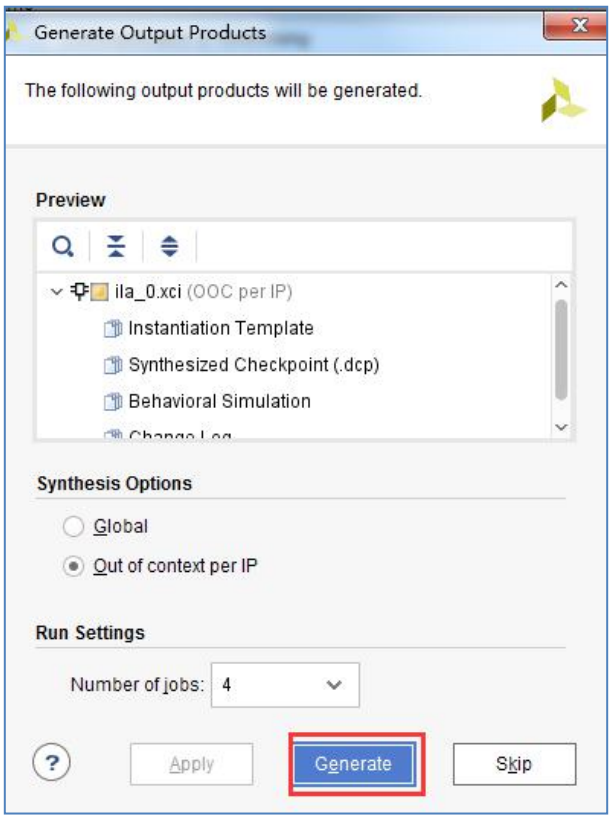
再对前面 8 个通道设置数据宽度，这里Probe width 根据接的信号位宽而定这里按照mem_test的端口进行设置



再设置后面 8~12通道的数据宽度



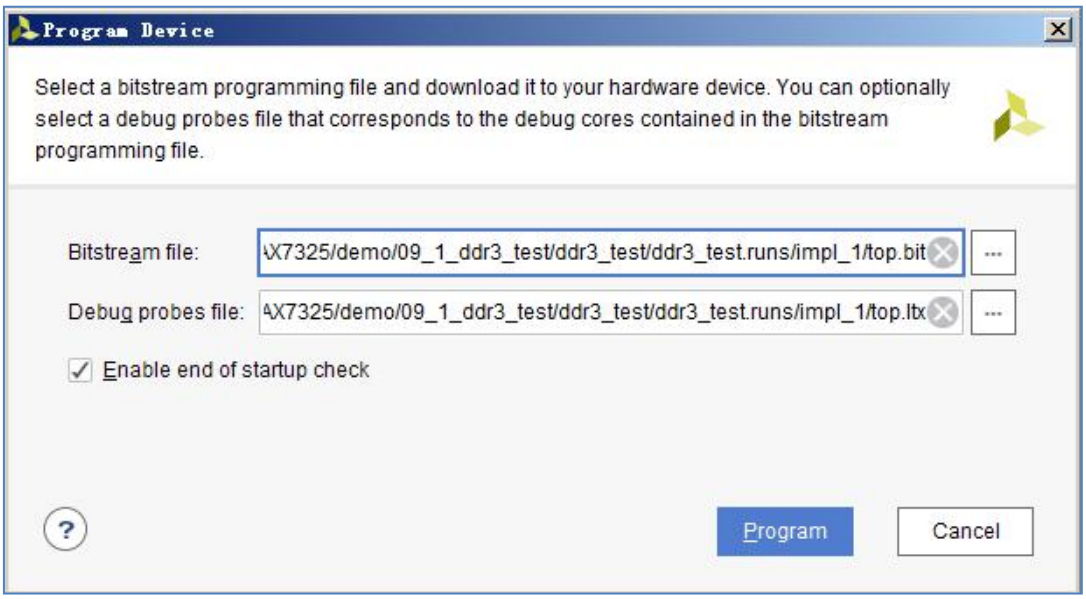
点击 Generate 按钮生成 IP 的设计文件。



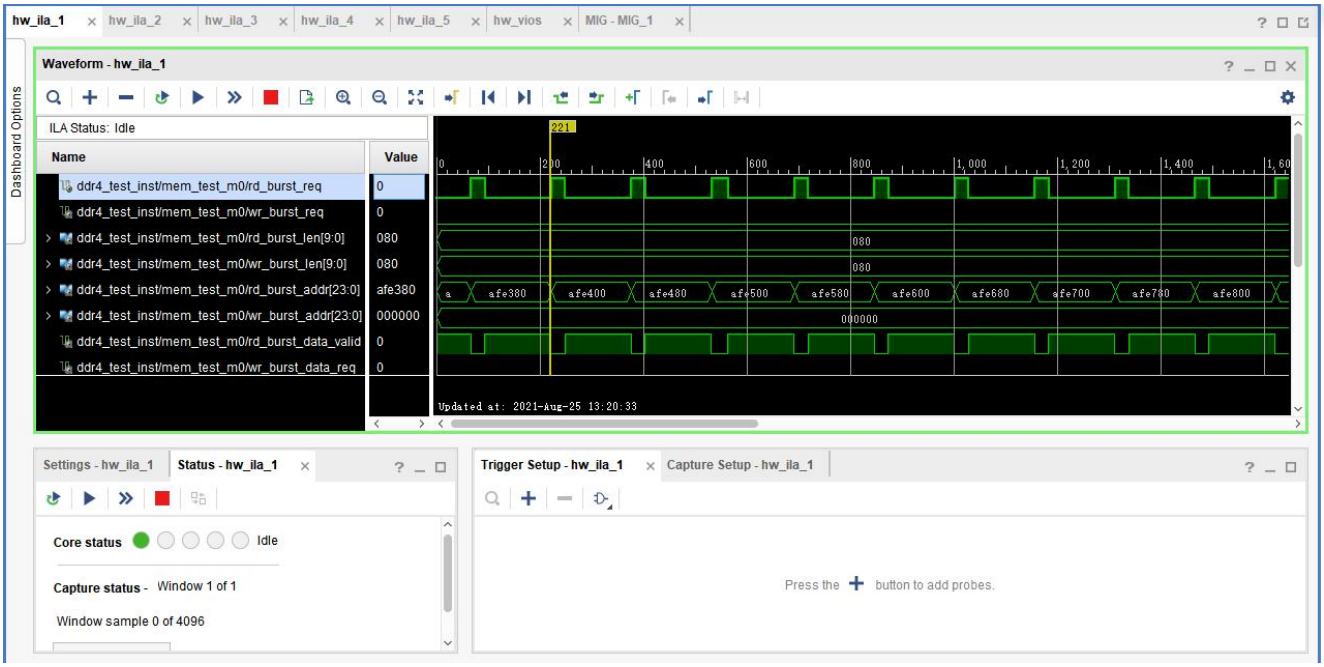
再来添加 xdc 管脚约束文件（见工程工文件），另外设置系统的复位信号 rst_n 和 FPGA 时钟输入sys_clk，fan_pwm 分配管脚控制风扇开启。

6 下载和测试

译程序生成 top.bit 文件，然后下载 bit 文件到 FPGA。

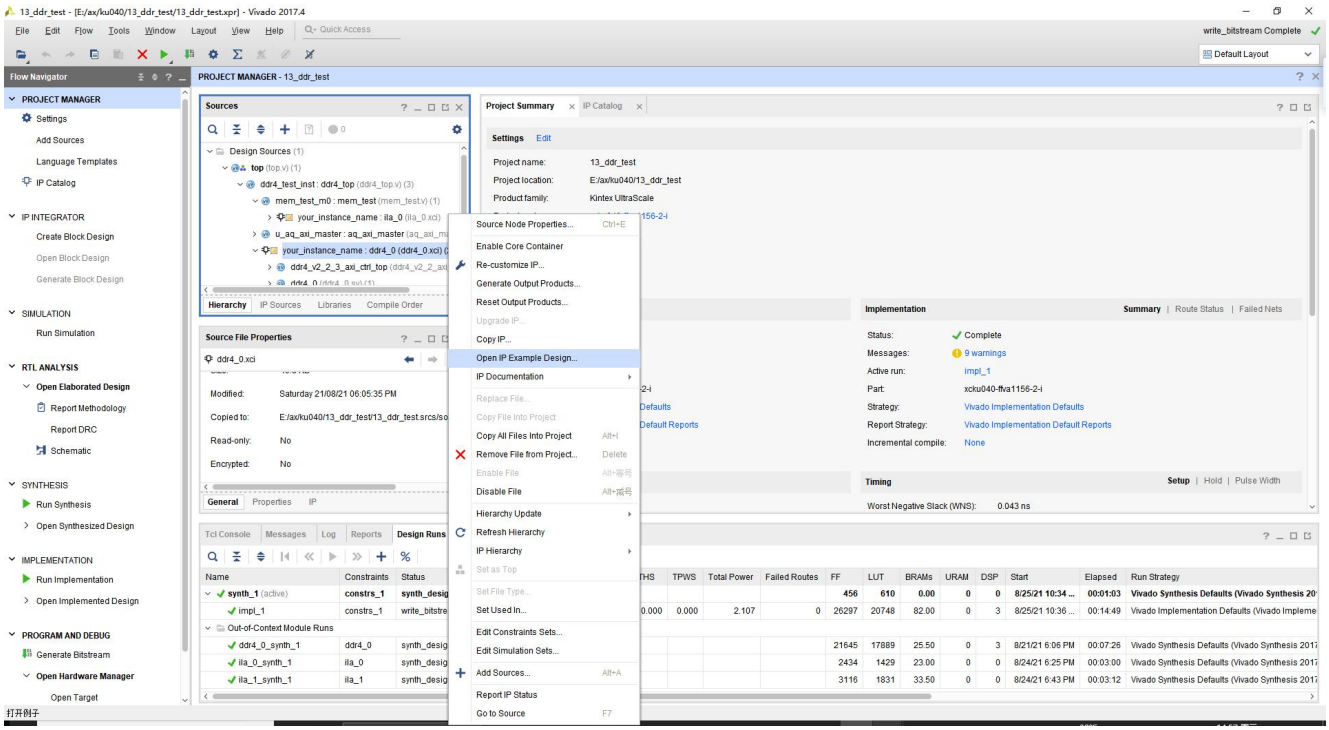


下载后软件会自动打开 ila 的波形调试界面，点击“Run trigger for ILA core” 按钮运行 ila 采集数据。

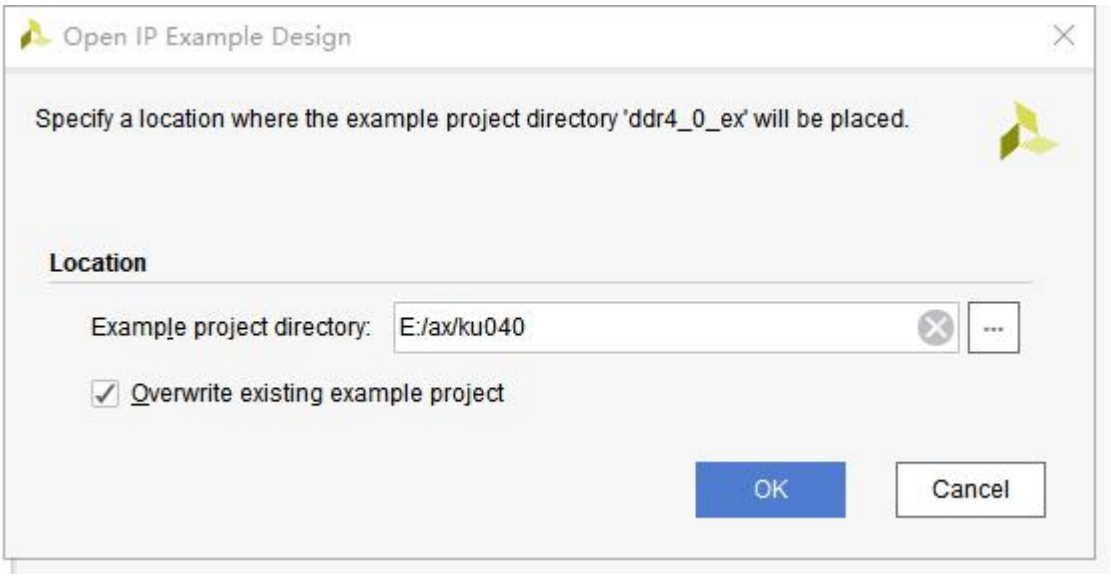


7 DDR4仿真

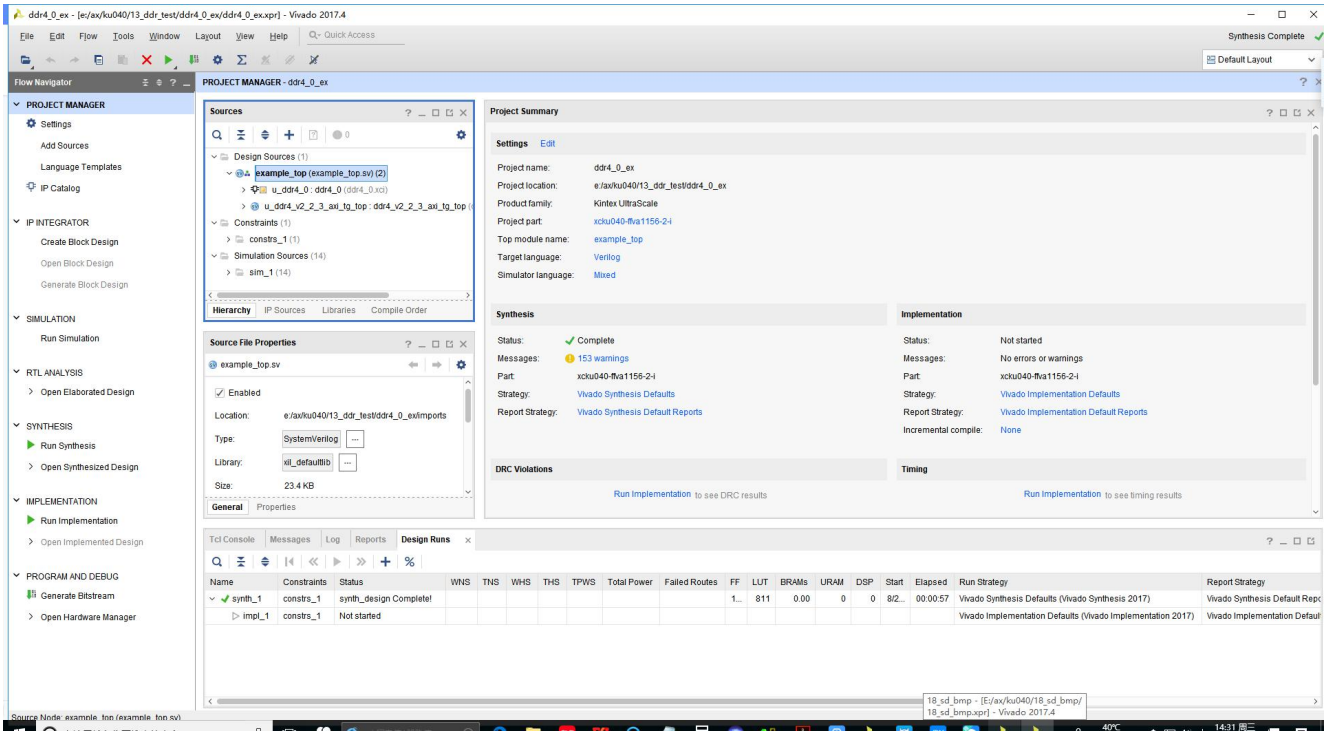
1. 在上面的例程中右键选择 ddr4 IP，在弹出的下拉菜单里选择 Open IP Example Design 。



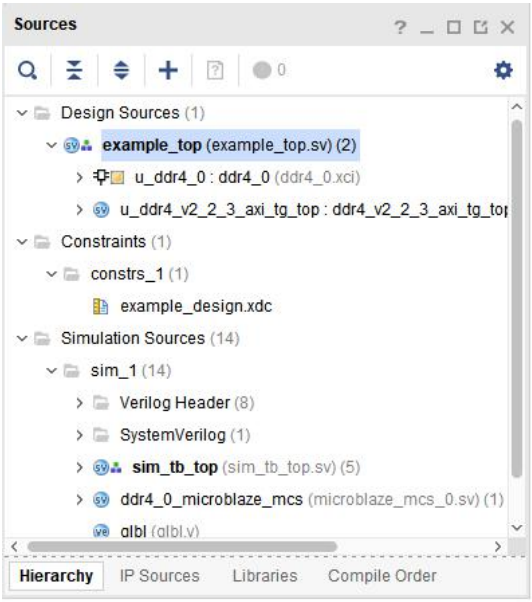
Ddr4_ex 项目生成的路径是选择默认，可以不用修改，点击 OK 。



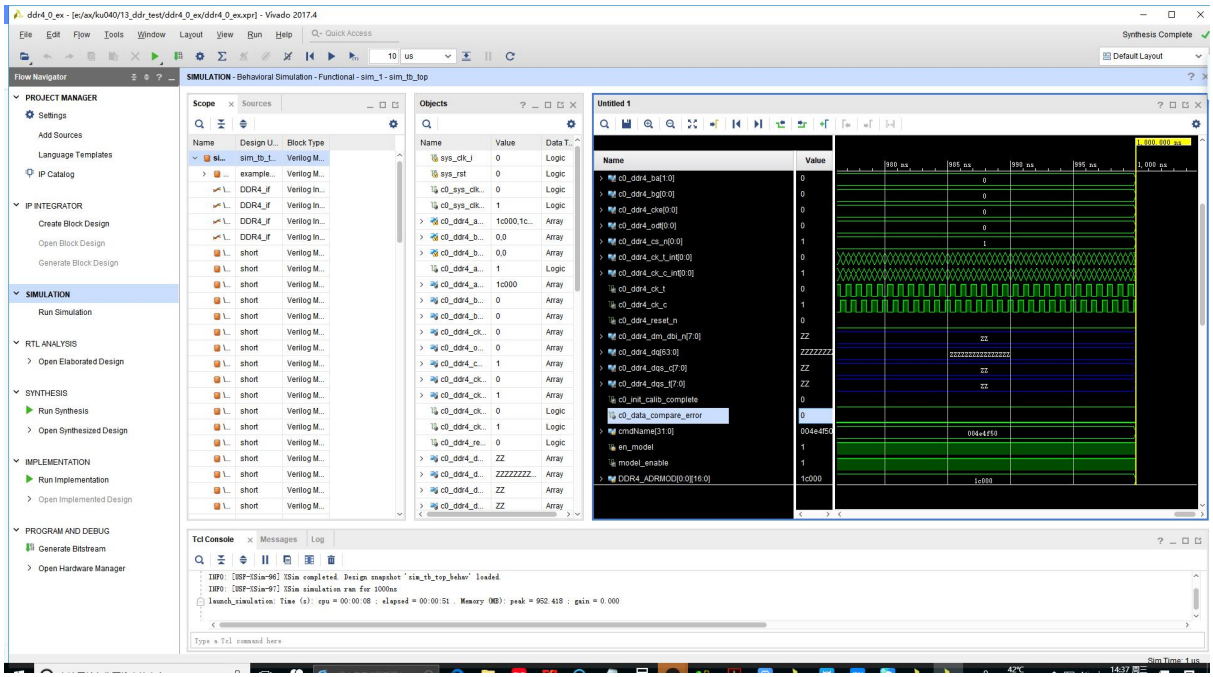
软件会自动打开生成的 ddr4_ex 工程如下图：

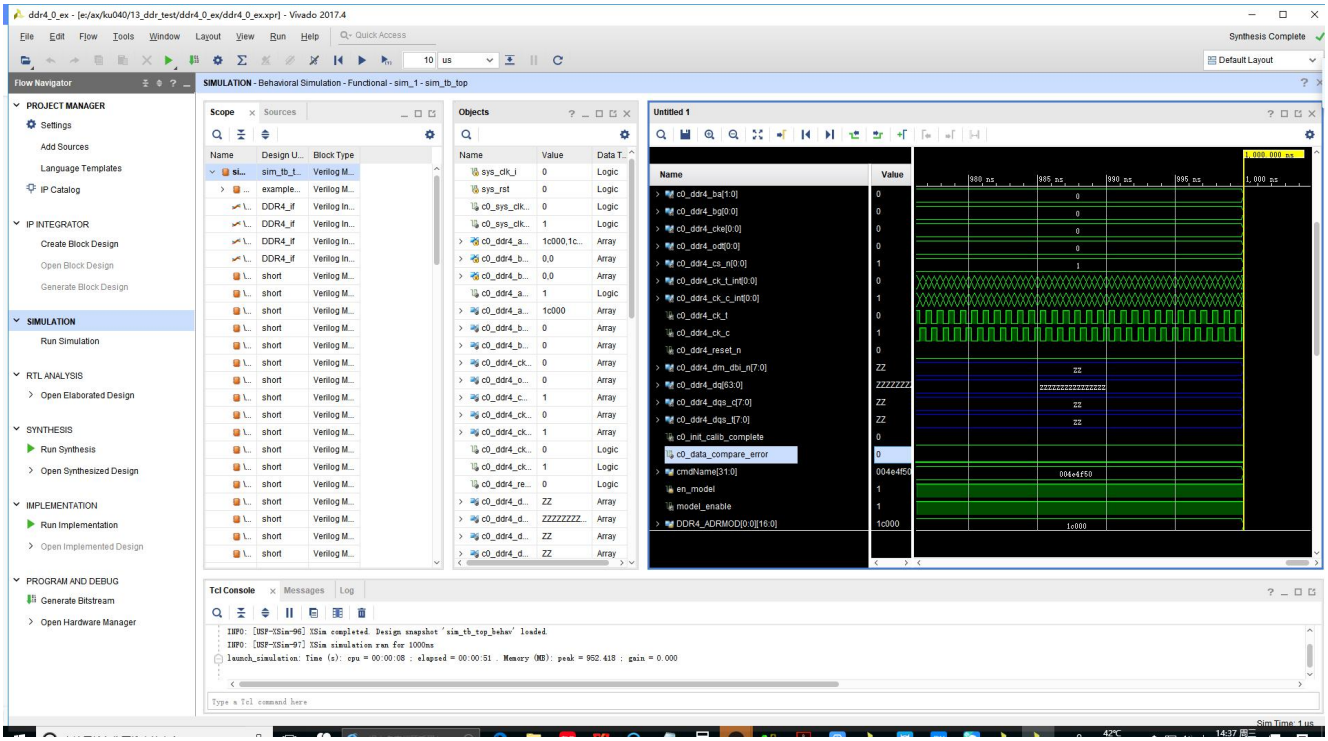


这个 ddr4_ex 工程里，软件已经自动编写了 仿真文件，xdc文件 ：



2. 点击 Run Simulation 按钮，再选择 Run Behavioral Simulation。这里我们做一下行为级的仿真





3. 仿真时间这里设置为10ms 运行 等待出现测试完成