MPSoC逻辑加速模块数据通道快速设计

作者： 付汉杰 hankf@xilinx.com

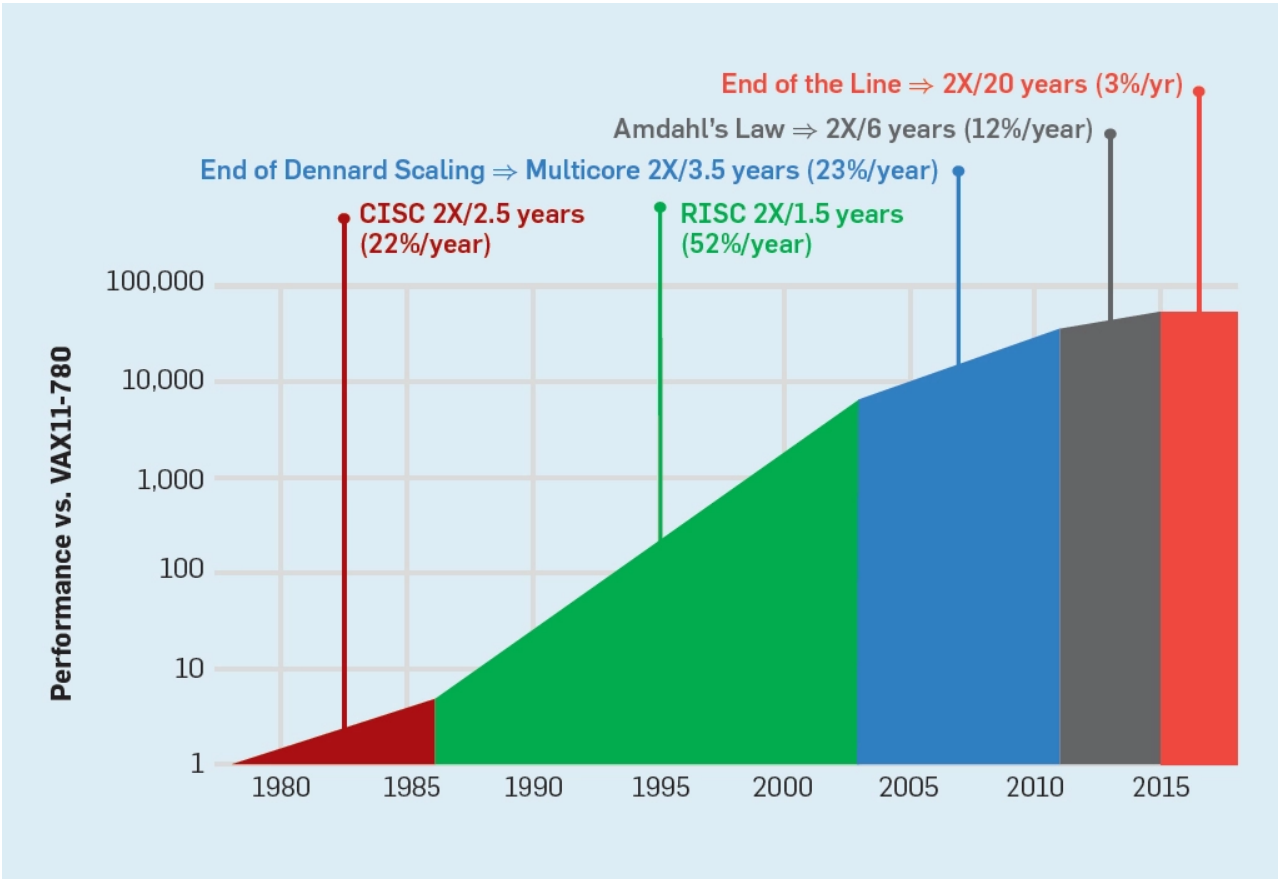
版本： 1.0

时间： 2019-11-13

回到目录前 ***** 回到目录后

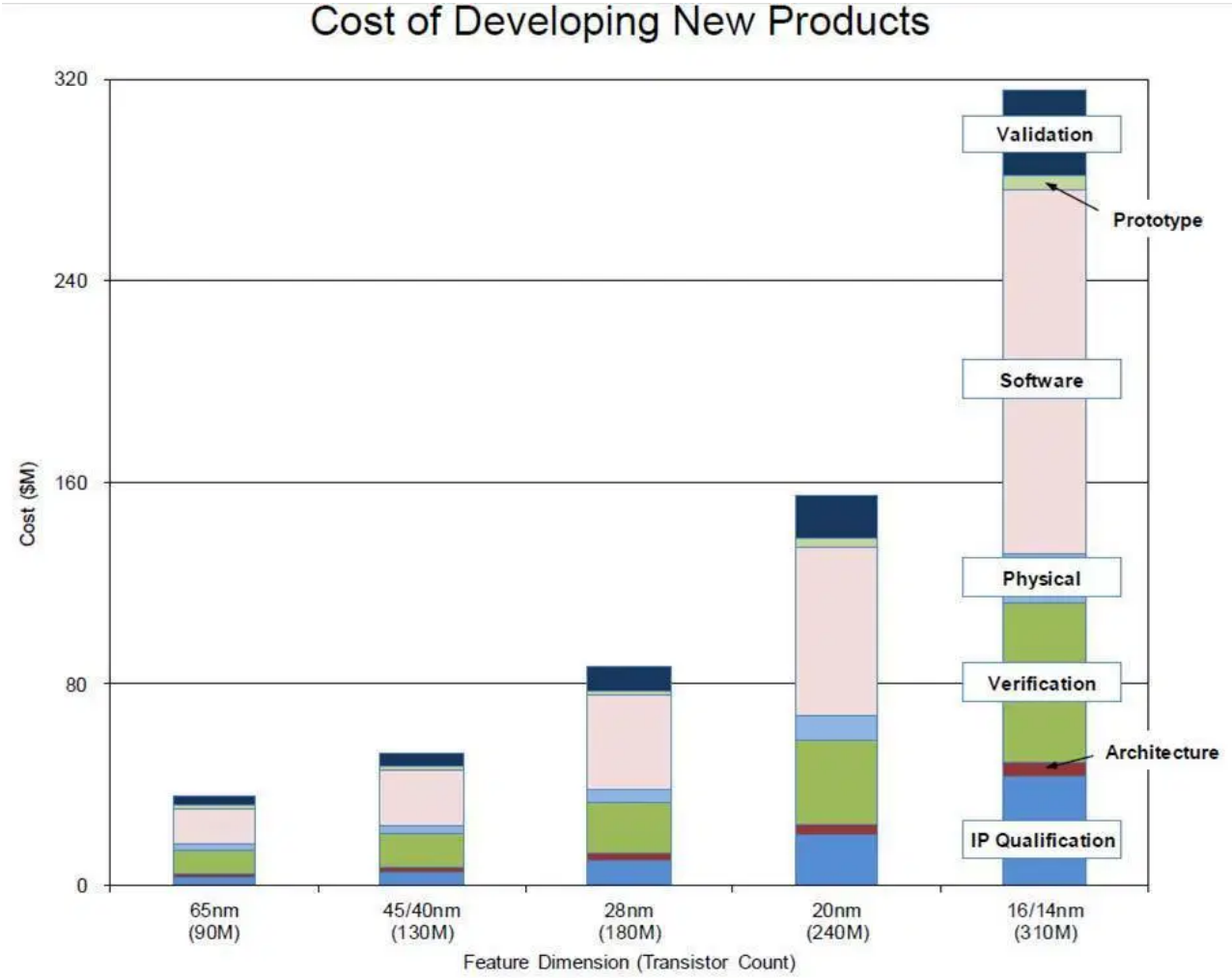
1. 概述

最近十几年，整个芯片产业都感觉到了摩尔定律的放缓，甚至失效。根据A Domain-Specific Architecture for Deep Neural Networks，以后CPU每年的性能只能进步3%左右。要改进系统性能，只能定制架构（Domain-Specific Architecture）和芯片。



图片来源于A Domain-Specific Architecture for Deep Neural Networks

而芯片设计的成本在不断攀升。即使是成熟的28nm工艺，预估的芯片设计成本也超过8千万美元。因此，定制架构和芯片不是普通中小企业能够使用的办法。



图片来源于FINFET AND FD SOI: MARKET AND COST ANALYSIS

Xilinx SoC芯片，是设计定制架构（Domain-Specific Architecture）的最实惠最可行的办法。这也是传统的视频应用和最前沿的机器学习应用，都大量用到Xilinx SoC芯片的原因。Xilinx SoC芯片，可以在PL（FPGA）中加速各种消耗CPU时间的算法。只要在PL里添加应用的加速算法，Xilinx SoC芯片就变成了Domain Specific Architecture，变成了ASIC(Application Specific Integrated Circuit)。

在PL（FPGA）里集成加速模块，需要为加速模块提供高速高效的数据通道。Xilinx为了方便客户设计，在芯片、IP、驱动层面为数据通道提供了高效可靠的可重用模块。

1.1. Xilinx SoC芯片对数据通道的支持

Xilinx SoC芯片里，在PS和PL之间，有多个高速数据通道。以MPSoC为例，PS（处理器子系统）和PL（FPGA）之间，PS作为主设备的通路有HPM0_FPD、HPM1_FPD、HPM0_LPD, 可以用于PS控制PL设备；PL作为主设备的通路有HP0、HP1、HP2、HP3、HPC0、HPC1、ACP、ACE, 可以用于PL访问PS的DDR，使PS和PL通过共享内存交互数据。特别值得一提的是，HPC0、HPC1、ACP、ACE具有cache同步能力。因此后续的例子中，使用了HPC端口，从而在一个硬件设计上，既可以使用软件管理cache同步，也可以使用硬件管理cache同步。

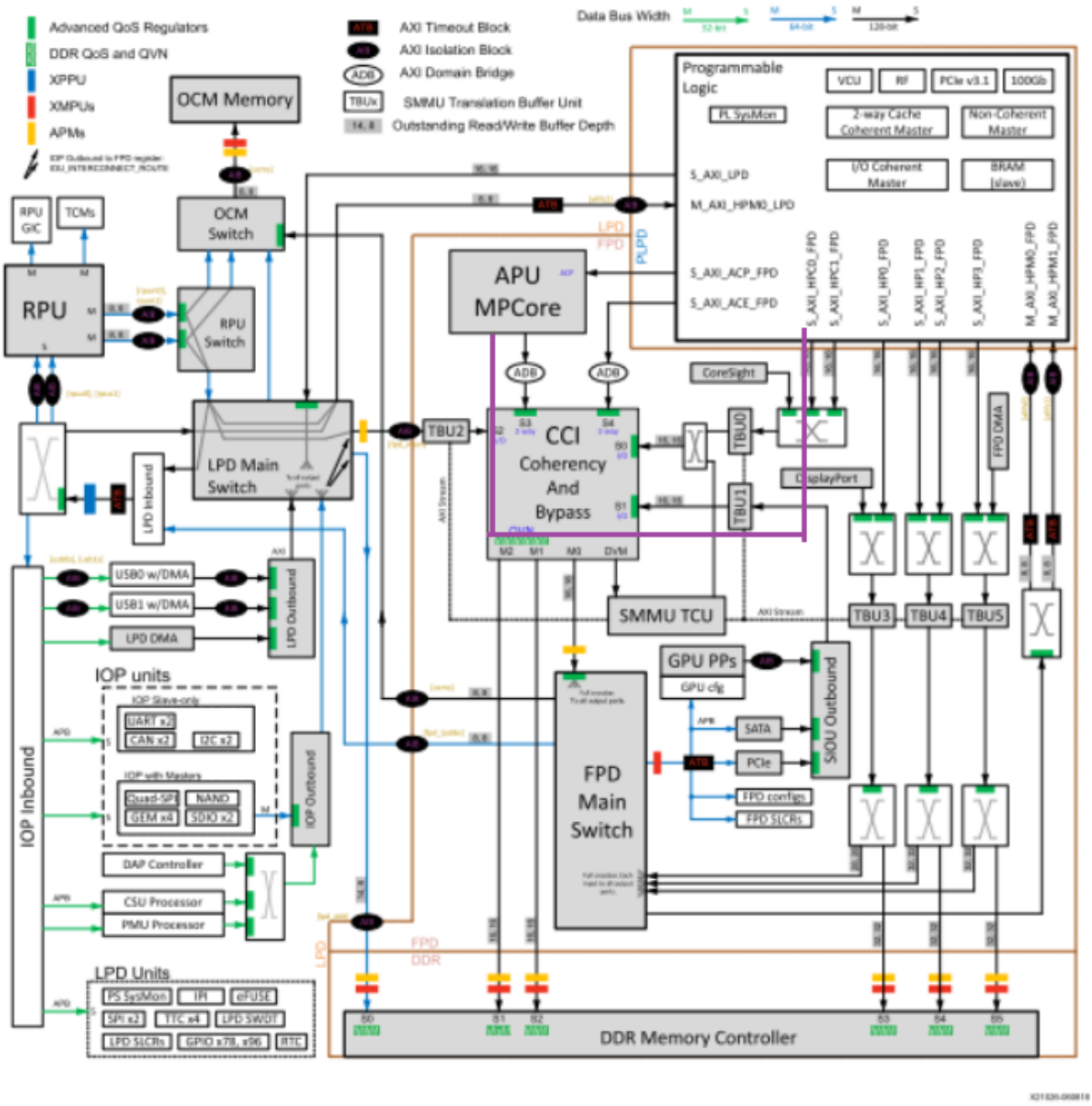


Figure 1-1: **AXI Interconnect**

图片来源于UG1085 Zynq UltraScale+ Device Technical Reference Manual v2.1 August 21, 2019

1.2. Xilinx IP对数据通道的支持

Xilinx提供了多个DMA IP, 比如AXI CDMA, AXI VDMA, AXI DMA。CDMA适合在内存之间搬移数据。VDMA主要用于搬移图像数据。AXI DMA的接口是AXI Stream接口，逻辑设计简单，效率高，最适合作为加速模块的数据通道。

1.3. Xilinx 驱动对数据通道的支持

Xilinx为AXI DMA提供成熟稳定的Linux驱动程序、standalone 驱动程序。AXI DMA Linux驱动程序，支持Linux DMA框架。如果在Linux内核使用AXI DMA，按Linux的DMA标准使用流程就行。更多信息，请参考 Xilinx SoftIP DMA'S Linux driver.

1.4. 应用程序对数据通道的支持

应用程序中对DMA的使用，与业务相关，千差万别，Xilinx没有提供一个标准应用程序。但是AXI DMA已经在业界得到了广泛的应用，有多个开源软件支持AXI DMA。
后续的设计中，使用Brandon Perez Xilinx AXI DMA Driver and Library。

回到目录前 ***** 回到目录后

2. 测试环境

后续所有设计使用下面的主机、单板和工具进行。

2.1. 主机

Ubuntu 16.04

2.2. 单板及其软件

1. ZCU106
2. U-Boot 2019.1
3. Linux Kernel 4.19

2.3. 工具

1. Vivado 2019.1
2. SDK 2019.1
3. PetaLinux 2019.1

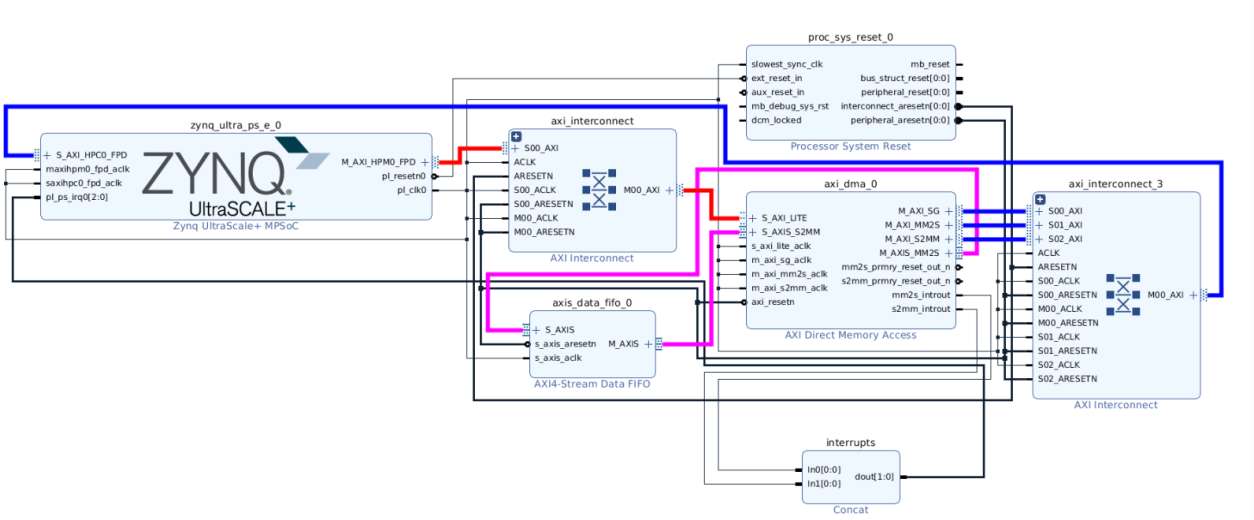
回到目录前 ***** 回到目录后

3. 软件同步Cache的数据通道设计

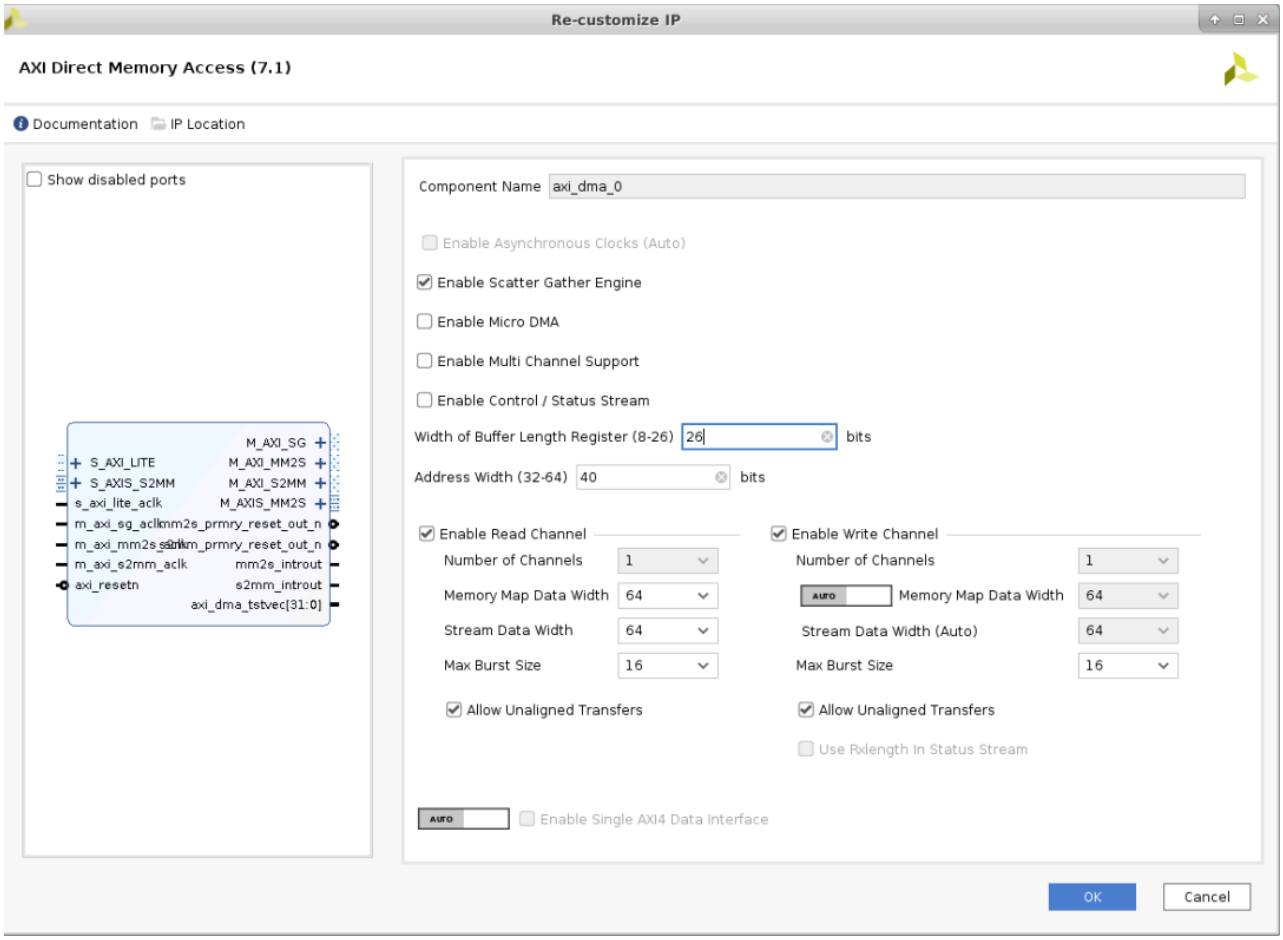
3.1. 软件同步Cache的数据通道的硬件设计

大多数嵌入式系统的数据通道，CPU软件负责管理cache同步。对于这种软件管理Cache的数据通道，设计简单，只需要把AXI DMA的管理端口S_AXI_LITE连接到PS的HPM0_FPD，使CPU能控制AXI DMA; 把各个M_AXI接口通过AXI Interconnect连接到HPC0_FPD端口，使AXI DMA能读写DDR；MM2S和S2MM通过AXIS_data_fifo连接到一起，实现数据传输；把中断输出通过IP Concat 连接到PS的中断输入。在实际应用中，以实际业务的模块替换AXIS_data_fifo。
设计中有两个注意事项。如果系统中有超过32-bit地址的DDR空间，建议让AXI DMA支持40-bit地址。Xilinx MPSoC超过2GB的DDR的地址是0x800000000（32GB）。如果AXI DMA不支持超过32-bit的地址，就不能访问在0x800000000的DDR，可能导致问题。AXI DMA的数据缓存比较小，如果数据量大，建议在数据通道上添加AXIS_data_fifo，避免数据溢出和丢失。
为了和后续设计保持一致，下面的设计使用了HPC端口。如果换成HP端口，也是可以工作的。

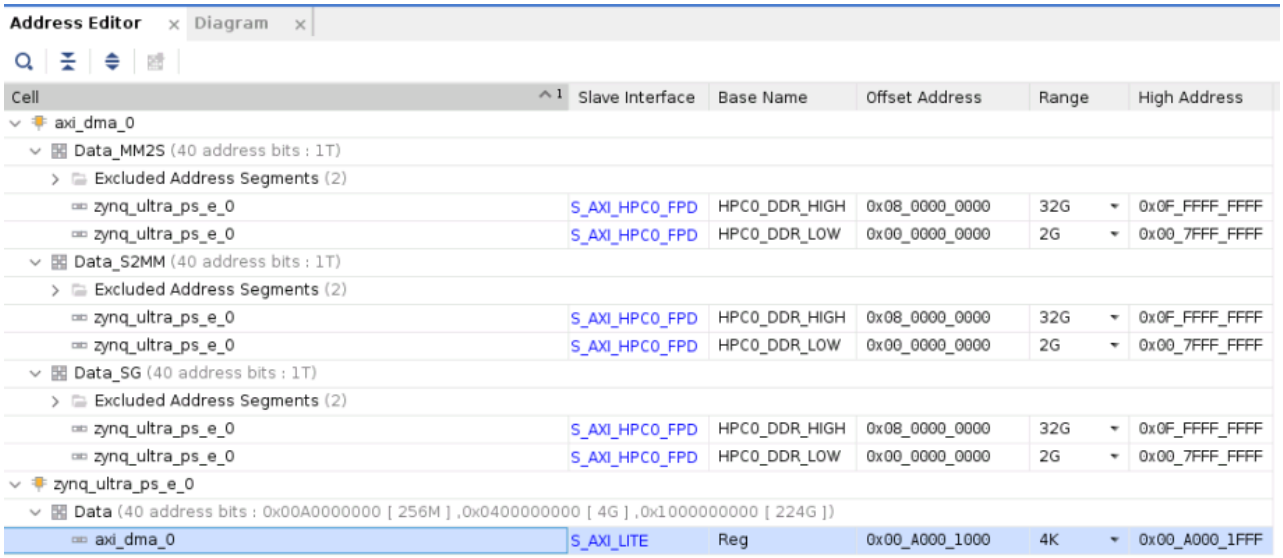
MPSoC AXI DMA连接图



AXI DMA内部配置



MPSoC AXI DMA设计地址分配图



回到目录前 ***** 回到目录后

3.2. 软件同步Cache的数据通道的Linux驱动设计

3.2.1. Linux Kernel选项

为了在Linux中使用AXI DMA, 需要在Linux Kernel中为其增加驱动程序、测试代码、以及一些辅助模块。请在Linux Kernel中，为其增加下列模块。配置后，也可以在Linux的配置文件.config可以找到下列项目。

```
1. CONFIG_DMADEVICES
2. CONFIG_XILINX_DMA
3. CONFIG_XILINX_DMATEST=m
4. CONFIG_ARM64_PTDUMP_CORE=y
5. CONFIG_ARM64_PTDUMP_DEBUGFS=y
```

3.2.2. Linux Devicetree

PetaLinux自动为AXI DMA在plnx_workspace\device-tree\device-tree\pl.dtsi创建了Devicetree节点。注意，AXI DMA的Devicetree节点的标签（label）被定义成axi_dma_0，它在后面会被反复引用。

```
    axi_dma_0: dma@a0001000 {
        #dma-cells = <1>;
        clock-names = "s_axi_lite_aclk", "m_axi_sg_aclk", "m_axi_mm2s_aclk",
"m_axi_s2mm_aclk";
        clocks = <&zynqmp_clk 71>, <&zynqmp_clk 71>, <&zynqmp_clk 71>, <&zynqmp_clk
71>;
        compatible = "xlnx,axi-dma-7.1", "xlnx,axi-dma-1.00.a";
        interrupt-names = "mm2s_introut", "s2mm_introut";
        interrupt-parent = <&gic>;
        interrupts = <0 89 4 0 90 4>;
        reg = <0x0 0xa0001000 0x0 0x1000>;
        xlnx,addrwidth = <0x28>;
        xlnx,include-sg ;
```

```
        xlnx,sg-length-width = <0x1a>;
        dma-channel@a0001000 {
            compatible = "xlnx,axi-dma-mm2s-channel";
            dma-channels = <0x1>;
            interrupts = <0 89 4>;
            xlnx,datawidth = <0x40>;
            xlnx,device-id = <0x0>;
            xlnx,include-dre ;
        };
        dma-channel@a0001030 {
            compatible = "xlnx,axi-dma-s2mm-channel";
            dma-channels = <0x1>;
            interrupts = <0 90 4>;
            xlnx,datawidth = <0x40>;
            xlnx,device-id = <0x0>;
            xlnx,include-dre ;
        };
    };
```

工程师需要使能AXI DMA的Devicetree节点，请在meta-user\recipes-bsp\device-tree\files\system-user.dtsi中为它添加属性status = "okay"。

```
&axi_dma_0 {
                status = "okay";
    };
```

有了上述Devicetree节点，AXI DMA的驱动程序能自动从系统得到自己的运行信息，比如地址、中断号等。

回到目录前 ***** 回到目录后

3.3. 软件同步Cache的数据通道的Linux内核测试

3.3.1. Linux内核测试模块

在内核测试AXI DMA，需要在内核配置中选择CONFIG_XILINX_DMATEST。前面的Linux Kernel选项 已经以模块形式选择CONFIG_XILINX_DMATEST。

编译后，单板的文件系统中会有测试模块的文件axidmatest.ko。

```
# ls /lib/modules/4.19.0-xilinx-v2019.1/kernel/drivers/dma/xilinx/ -l
total 20
-rw-r--r--    1 root     root         19760 Nov  1 08:01 axidmatest.ko
```

3.3.2. Linux内核测试的Devicetree

在内核测试AXI DMA，要在Devicetree中指定测试模块使用的AXI DMA的模块名和通道号。下面的devicetree, 测试模块使用"dmas = <&axi_dma_0 0 &axi_dma_0 1>"指定了设备的标签axi_dma_0，表示测试模块使用Linux Devicetree中定义的在地址a0001000的AXI DMA设备。
如果硬件设计中有多个AXI DMA，可以指定不同AXI DMA的label，测试不同的AXI DMA。注意，axidmatest.c的函数xilinx_axidmatest_probe()调用dma_request_slave_channel()时，使用了dma-names之后的字符串axidma0和axidma1，因此在Devicetree文件也必须使用字符串axidma0和axidma1，不能

使用其它的名称。其中axidma0在axidmatest.c的被作为mm2s的DMA通道，axidma1被作为s2mm的DMA通道。

```
/ {
    axidmatest_1: axidmatest@1 {
        compatible ="xlnx,axi-dma-test-1.00.a";
       dmas = <&axi_dma_0 0
               &axi_dma_0 1>;
        dma-names = "axidma0", "axidma1";
               } ;
};
```

内核测试模块axidmatest.c中使用字符串axidma0和axidma1的代码如下：

```
static int xilinx_axidmatest_probe(struct platform_device *pdev)
{
    struct dma_chan *chan, *rx_chan;
    int err;

    chan = dma_request_slave_channel(&pdev->dev, "axidma0");
    if (IS_ERR(chan)) {
        pr_err("xilinx_dmatest: No Tx channel\n");
        return PTR_ERR(chan);
    }

    rx_chan = dma_request_slave_channel(&pdev->dev, "axidma1");
    if (IS_ERR(rx_chan)) {
        err = PTR_ERR(rx_chan);
        pr_err("xilinx_dmatest: No Rx channel\n");
        goto free_tx;
    }

    err = dmatest_add_slave_channels(chan, rx_chan);
    if (err) {
        pr_err("xilinx_dmatest: Unable to add channels\n");
        goto free_rx;
    }

    return 0;

free_rx:
    dma_release_channel(rx_chan);
free_tx:
    dma_release_channel(chan);

    return err;
}
```

3.3.3. Linux内核测试

加载内核测试模块的文件axidmatest.ko，会启动测试，利用AXI DMA的mm2s通道把数据从DDR搬移到AXI Stream接口，再利用AXI DMA的s2mm DMA通道把数据搬移到DDR。测试次的记录如下：

```
# insmod axidmatest.ko
[  210.871166] dmatest: Started 1 threads using dma1chan0 dma1chan1
[  212.937889] dma1chan0-dma1c: terminating after 5 tests, 0 failures (status 0)
```

内核测试模块axidmatest.ko还支持参数，test_buf_size指定测试内存块的字节数，缺省是16384字节；iterations指定测试次数，缺省是5次。测试100次的记录如下：

```
# insmod axidmatest.ko test_buf_size=65536 iterations=100
[  210.871166] dmatest: Started 1 threads using dma1chan0 dma1chan1
[  212.937889] dma1chan0-dma1c: terminating after 100 tests, 0 failures (status 0)
```

回到目录前 ***** 回到目录后

3.4. 软件同步Cache的数据通道的应用程序测试

3.4.1. 应用程序

为了方便，使用Brandon Perez在Github上开源的驱动和应用程序 Xilinx AXI DMA Driver and Library测试。
测试代码包含一个DMA上层驱动，给DMA创建设备节点；数个应用程序，其中axidma_benchmark可以测试AXI DMA性能。

3.4.2. 应用程序的Devicetree

测试之前，通过Devicetree指定DMA上层驱动测试时使用的AXI DMA设备。对于下面的devicetree,DMA上层驱动使用"dmas = <&axi_dma_0 0 &axi_dma_0 1>"指定了设备的标签axi_dma_0，表示DMA上层驱动使用地址a0001000的AXI DMA设备。
测试中，AXI DMA设备有两个通道，一个是mm2s，一个是s2mm。Xilinx PetaLinux生成的设备树中，两个通道的device-id都是0。而Brandon Perez的DMA上层驱动要求每个通道的device-id不同。因此在devicetree中，使用delete-node删除了自动生成的设备节点axi_dma_0，并复制原来的内容，把其中的device-id分别改为了1和2。

```
/delete-node/ &axi_dma_0;

/ {
    axi_dma_0: dma@a0001000 {
        #dma-cells = <1>;
        clock-names = "s_axi_lite_aclk", "m_axi_sg_aclk", "m_axi_mm2s_aclk",
"m_axi_s2mm_aclk";
        clocks = <&zynqmp_clk 71>, <&zynqmp_clk 71>, <&zynqmp_clk 71>, <&zynqmp_clk
71>;
        compatible = "xlnx,axi-dma-7.1", "xlnx,axi-dma-1.00.a";
        interrupt-names = "mm2s_introut", "s2mm_introut";
        interrupt-parent = <&gic>;
        interrupts = <0 89 4 0 90 4>;
        reg = <0x0 0xa0001000 0x0 0x1000>;
        xlnx,addrwidth = <0x28>;
```

```
        xlnx,include-sg ;
        xlnx,sg-length-width = <0x1a>;
        dma-channel@a0001000 {
            compatible = "xlnx,axi-dma-mm2s-channel";
            dma-channels = <0x1>;
            interrupts = <0 89 4>;
            xlnx,datawidth = <0x40>;
            xlnx,device-id = <0x1>;
            xlnx,include-dre ;
        };
        dma-channel@a0001030 {
            compatible = "xlnx,axi-dma-s2mm-channel";
            dma-channels = <0x1>;
            interrupts = <0 90 4>;
            xlnx,datawidth = <0x40>;
            xlnx,device-id = <0x2>;
            xlnx,include-dre ;
        };
    };

    axidma_chrdev: axidma_chrdev@0 {
        compatible = "xlnx,axidma-chrdev";
        dmas = <&axi_dma_0 0 &axi_dma_0 1>;
        dma-names = "tx_channel", "rx_channel";
    };
};
```

3.4.3. 应用程序的修改

由于Linux Kernel版本变化，需要修改部分代码。

在driver\axidma_dma.c里，需要增加头文件of_dma.h。否则，会为axidma_compatible_of_ids[] 报告错误“field name not in record or union initializer”。

```
#include <linux/of_dma.h>
```

在driver\axidma_chrdev.c的函数axidma_mmap()里，需要为of_dma_configure增加一个参数。否则，会为of_dma_configure()报告错误“error: too few arguments to function of_dma_configure”。

```
    // Configure the DMA device
    //of_dma_configure(dev->device, NULL);
    of_dma_configure(dev->device, NULL, true);
```

回到目录前 ***** 回到目录后

3.4.4. 应用程序编译

在X86的机器上编译MPSoC A53的代码，需要通过CROSS_COMPILE设置交叉编译器。编译驱动代码，需要通过ARCH设置CPU架构，并且知道Linux Kernel的源代码。编译完成后，在目录outputs下，有DMA上层驱动文件axidma.ko，和测试应用程序axidma_benchmark。

编译命令：

```
$ export ARCH=arm64
$ export CROSS_COMPILE=aarch64-linux-gnu-
$ make KBUILD_DIR=../source/linux-kernel/ clean
$ make KBUILD_DIR=../source/linux-kernel/
```

3.4.4.1. 应用程序编译的完整日志：

编译的完整日志

回到目录前 ***** 回到目录后

3.4.5. 应用程序测试

将目录outputs下的所有文件，复制到单板上，增加可执行权限，再安装DMA上层驱动文件axidma.ko，最后执行测试应用程序axidma_benchmark。

```
root@xilinx-zcu106-2019_1:~# insmod axidma.ko
[  133.465535] axidma: axidma_dma.c: axidma_dma_init: 729: DMA: Found 1 transmit
channels and 1 receive channels.
[  133.475533] axidma: axidma_dma.c: axidma_dma_init: 731: VDMA: Found 0 transmit
channels and 0 receive channels.

root@xilinx-zcu106-2019_1:~# ./axidma_benchmark
AXI DMA Benchmark Parameters:
        Transmit Buffer Size: 7.91 MiB
        R[  218.929479] axidma axidma: DMA mask not set
eceive Buffer Size: 7.91 MiB
        Number of DMA Transfers: 1000 transfers
Using transmit channel 1 and receive channel 2.
Single transfer test successfully completed!
Beginning performance analysis of the DMA engine.

DMA Timing Statistics:
        Elapsed Time: 10.41 s
        Transmit Throughput: 760.01 MiB/s
        Receive Throughput: 760.01 MiB/s
        Total Throughput: 1520.03 MiB/s
```

可以看到，AXI DMA能为发送和接收双方向都提供760MiBps的性能。

3.4.5.1. 应用程序测试的完整记录

回到目录前 ***** 回到目录后

3.4.6. 双功能的Linux Devicetree

Devicetree只是提供一些参数。我们不会同时运行内核测试和应用程序测试。因此可以内核测试和应用程序测试的参数同时放在一个Devicetree里，是一个内核文件既支持内核测试，又支持应用程序测试。

为了便于读者测试，下面提供统一的Linux Devicetree。它在应用程序的Devicetree基础上，增加了内核测试模块的参数。

```
/delete-node/ &axi_dma_0;

/ {
    chosen {
              bootargs = "cpuidle.off=1";
       };

    axi_dma_0: dma@a0001000 {
        #dma-cells = <1>;
        clock-names = "s_axi_lite_aclk", "m_axi_sg_aclk", "m_axi_mm2s_aclk",
"m_axi_s2mm_aclk";
        clocks = <&zynqmp_clk 71>, <&zynqmp_clk 71>, <&zynqmp_clk 71>, <&zynqmp_clk
71>;
        compatible = "xlnx,axi-dma-7.1", "xlnx,axi-dma-1.00.a";
        interrupt-names = "mm2s_introut", "s2mm_introut";
        interrupt-parent = <&gic>;
        interrupts = <0 89 4 0 90 4>;
        reg = <0x0 0xa0001000 0x0 0x1000>;
        xlnx,addrwidth = <0x28>;
        xlnx,include-sg ;
        xlnx,sg-length-width = <0x1a>;
        status = "okay";

        dma-channel@a0001000 {
            compatible = "xlnx,axi-dma-mm2s-channel";
            dma-channels = <0x1>;
            interrupts = <0 89 4>;
            xlnx,datawidth = <0x40>;
            xlnx,device-id = <0x1>;
            xlnx,include-dre ;
        };
        dma-channel@a0001030 {
            compatible = "xlnx,axi-dma-s2mm-channel";
            dma-channels = <0x1>;
            interrupts = <0 90 4>;
            xlnx,datawidth = <0x40>;
            xlnx,device-id = <0x2>;
            xlnx,include-dre ;
        };
    };

    axidmatest_1: axidmatest@1 {
            compatible ="xlnx,axi-dma-test-1.00.a";
               dmas = <&axi_dma_0 0
                       &axi_dma_0 1>;
            dma-names = "axidma0", "axidma1";
    } ;

    axidma_chrdev: axidma_chrdev@0 {
        compatible = "xlnx,axidma-chrdev";
        dmas = <&axi_dma_0 0 &axi_dma_0 1>;
        dma-names = "tx_channel", "rx_channel";
```

```
    };
};
```

回到目录前 ***** 回到目录后

3.4.7. 单板上的实际Devicetree

开发人员调试时，经常有多个不同属性的版本。Cache同步的设置，影响实际功能运行。如果不放心，建议在单板上检查，是否为DMA设置了硬件同步Cache功能。

在Linux单板上，Devicetree在单板的目录/sys/firmware/devicetree/base/下。

在下面的记录中，使用命令ls /sys/firmware/devicetree/base/dma@a0001000/查看DMA设备的信息。可以看到，地址a0001000为dma含有dma-coherent属性，支持硬件同步Cache功能。查看信息时，使用grep以关键词coherent过滤，可以排除其它信息。

```
root@xilinx-zcu106-2019_1:~# ls /proc/device-tree -l
lrwxrwxrwx    1 root     root            29 Nov 13 03:12 /proc/device-tree ->
/sys/firmware/devicetree/base

root@xilinx-zcu106-2019_1:~# ls /sys/firmware/devicetree/base/
#address-cells   aux_ref_clk      cpus             fclk1            gt_crx_ref_clk
pcap             smmu@fd800000
#size-cells      axidma_chrdev@0  dcc              fclk2            leds
pmu              timer
aliases          axidmatest@1     dma@a0001000     fclk3            memory
psci             video_clk
amba             chosen           dp_aclk          firmware         model
pss_alt_ref_clk  zynqmp_aes
amba_apu@0       compatible       edac             fpga-full        name
pss_ref_clk      zynqmp_ipi
amba_pl@0        cpu_opp_table    fclk0            gpio-keys        nvmem_firmware
sha384           zynqmp_rsa

root@xilinx-zcu106-2019_1:~# ls /sys/firmware/devicetree/base/dma\@a0001000/ -l
total 0
-r--r--r--    1 root     root             4 Nov 13 03:13 #dma-cells
-r--r--r--    1 root     root            62 Nov 13 03:13 clock-names
-r--r--r--    1 root     root            32 Nov 13 03:13 clocks
-r--r--r--    1 root     root            37 Nov 13 03:13 compatible
drwxr-xr-x    2 root     root             0 Nov 13 03:13 dma-channel@a0001000
drwxr-xr-x    2 root     root             0 Nov 13 03:13 dma-channel@a0001030
-r--r--r--    1 root     root             0 Nov 13 03:13 dma-coherent
-r--r--r--    1 root     root            26 Nov 13 03:13 interrupt-names
-r--r--r--    1 root     root             4 Nov 13 03:13 interrupt-parent
-r--r--r--    1 root     root            24 Nov 13 03:13 interrupts
-r--r--r--    1 root     root             4 Nov 13 03:13 name
-r--r--r--    1 root     root             4 Nov 13 03:13 phandle
-r--r--r--    1 root     root            16 Nov 13 03:13 reg
-r--r--r--    1 root     root             5 Nov 13 03:13 status
-r--r--r--    1 root     root             4 Nov 13 03:13 xlnx,addrwidth
-r--r--r--    1 root     root             0 Nov 13 03:13 xlnx,include-sg
-r--r--r--    1 root     root             4 Nov 13 03:13 xlnx,sg-length-width

root@xilinx-zcu106-2019_1:~# ls /sys/firmware/devicetree/base/dma\@a0001000/ -l |
```

```
grep coherent
-r--r--r--    1 root     root             0 Nov 13 06:36 dma-coherent
```

回到目录前 ***** 回到目录后

4. 硬件同步Cache的数据通道设计

MPSoC PL与PS之间的HPC0、HPC1、ACP、ACE端口，具有硬件同步cache能力。如果使用硬件管理cache同步，可以降低软件负担，提升系统能力。
现在使用HPC端口，建立硬件同步cache的数据通道。

4.1. AXI 信号

HPC端口也是AXI端口，含有所有AXI端口的信号。AXI主设备驱动的AxCACHE和AxPROT信号， 要满足相关的协议要求。具体情况请参考AMBA AXI and ACE Protocol Specification, ARM IHI 0022E。

4.1.1. AxCACHE

AXI上的AxCACHE，也就是ARCACHE[3:0]和AWCACHE[3:0]，描述AXI传输中的内存属性和cache控制属性。
需要把AXI上的ARCACHE[3:0]和AWCACHE[3:0]，设置成1111， 表示Write-back Read and Write-allocate。

4.1.2. AxPROT

AXI上的AxPROT，也就是ARPROT[2:0]和AWPROT[2:0]，表示AXI传输中的访问权限属性。如果CPU软件处于EL3，比如裸核(Standalone, bare metal)程序, 执行的是安全访问(secure access), AxPROT[1] 要设置成0。如果CPU软件处于EL1和EL0，比如Linux内核驱动和应用程序, 执行的是非安全访问(non-secure access), AxPROT[1] 要设置成1。

4.2. Cache监听（Snooping）

4.2.1. Broadcasting Inner Shareable

内存的cache属性包括non-shareable, inner shareable，outer shareable。inner shareable范围只包含A53 和L2 cache。缺省情况下，Inner Shareable的内存传输不会被广播给CCI。Linux缺省设置cacheable memory为Inner Shareable，导致CCI看不到所有cacheable memory的传输。为了使能硬件同步cache，必须使Inner Shareable的内存传输也被广播给CCI。
寄存器lpd_apu（0xFF41A040）的最低两位有这个功能，bit 0表示Enable broadcasting of Outer Shareable trasnactions；bit 1表示Enable broadcasting of Inner Shareable transactions。
注意，必须在A53处于复位态时设置寄存器lpd_apu。用启动文件boot.bin里的寄存器初始化功能设置寄存器lpd_apu，使BootROM在加载时设置寄存器lpd_apu，可以达到上述要求。

4.2.1.1. 寄存器初始化文件

准备寄存器初始化文件，置寄存器lpd_apu最低两位都为1。

```
.set. 0xFF41A040 = 0x3;
```

4.2.1.2. 修改启动文件boot.bin的配置信息

在启动文件boot.bin的配置信息中，以关键字"[init]"添加寄存器初始化文件，以创建含有寄存器初始化文件的启动文件。

```
//arch = zynqmp; split = false; format = BIN
the_ROM_image:
{
   ...
   [init]<path>\regs.init
}
```

完整的boot.bin的配置信息，bootgen.bif

```
/* bootgen -arch zynqmp -image bootgen.bif -o BOOT.BIN -w on */

the_ROM_image:
{
    [bootloader, destination_cpu=a53-0] ./zcu106_fsbl.elf
    [pmufw_image] ./images/linux/pmufw.elf
    [destination_device=pl] ./images/linux/system.bit
    [destination_cpu=a53-0, exception_level=el-3, trustzone] ./images/linux/bl31.elf
    [destination_cpu=a53-0, exception_level=el-2] ./images/linux/u-boot.elf
    [init]./regs.init
}
```

4.2.1.3. 创建启动文件boot.bin

SDK和PetaLinux都含有工具bootgen。
通过工具bootgen创建启动文件boot.bin的命令执行记录：

```
$ source /opt/Xilinx/peta/2019.1/settings.sh
$ which bootgen
/xilinxtool/peta/2019.1/tools/xsct/bin/bootgen
$ bootgen -arch zynqmp -image bootgen.bif -o BOOT.BIN -w on

********Xilinx Bootgen v2019.1
  ******Build date : May 24 2019-14:54:05
    ** Copyright 1986-2019 Xilinx, Inc. All Rights Reserved.
```

4.2.2. FSBL使能CCI监听APU

缺省情况下，MPSoC CCI （Cache Coherent Interconnect）不监听传输。为了利用CCI cache同步功能，需要使能CCI监听传输功能。

设置寄存器Snoop_Control_Register_S3(0xFD6E4000)的最低位，使CCI监听APU cluster。寄存器Snoop_Control_Register_S3最低位的含义是Enable issuing of snoop requests from slave interface S3。
在FSBL的xfsbl_main.c的main()里添加下列修改Snoop_Control_Register_S3的部分代码：

```
    case XFSBL_STAGE4:
    {
```

```
        XFsbl_Printf(DEBUG_INFO,
                "================== In Stage 4 ============ \n\r");

        {
            /* Modify Register Snoop_Control_Register_S3 for HPC Cache Cohenercy */
            unsigned int ui_snoop_control=0;
            XFsbl_Printf(DEBUG_PRINT_ALWAYS,"Check snoop control register at
0xfd6e4000.\n\r");
            ui_snoop_control = XFsbl_In32(0xfd6e4000);
            XFsbl_Printf(DEBUG_PRINT_ALWAYS,"Snoop control register at 0xfd6e4000
original value: 0x%08x.\n\r", ui_snoop_control );
            XFsbl_Out32(0xfd6e4000, ui_snoop_control|0x1);

            ui_snoop_control = XFsbl_In32(0xfd6e4000);
            XFsbl_Printf(DEBUG_PRINT_ALWAYS,"Snoop control register at 0xfd6e4000
new value: 0x%08x.\n\r", ui_snoop_control );
        }

        /**
         * Handoff to the applications
         * Handoff address
         * xip
         * ps7 post config
         */
        FsblStatus = XFsbl_Handoff(&FsblInstance, PartitionNum, EarlyHandoff);

        //... ...
    }
```

曾经尝试让寄存器初始化文件设置寄存器Snoop_Control_Register_S3，也就是下面的寄存器初始化文件，导致系统不能启动。

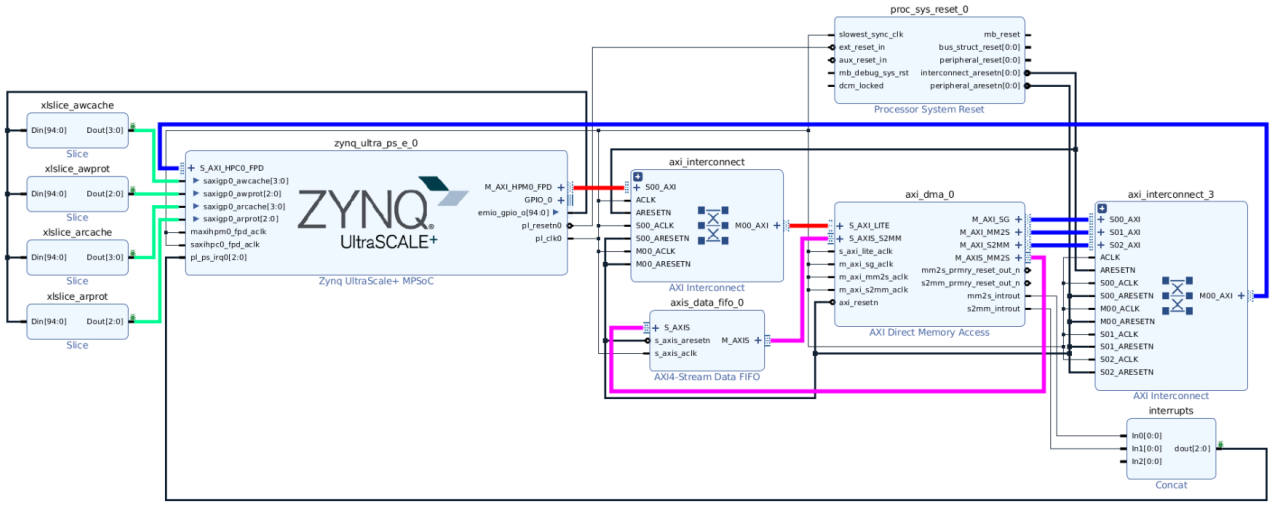
```
.set. 0xFF41A040 = 0x3;
.set. 0xFD6E4000 = 0xC0000001;
```

回到目录前 ***** 回到目录后

4.3. 硬件同步Cache的数据通道的硬件设计

在前面的软件管理Cache的数据通道的硬件设计的基础上，引出GPIO的EMIO管脚，通过Slice IP分配，再控制Cache相关的AXI信号。S_AXI_HPC0_FPD的AWCACHE[3:0]连接到了EMIO[3:0], AWPROT[2:0]连接到了EMIO[7:5],、ARCACHE[3:0]连接到了EMIO[11:8],ARPROT[2:0]信号连接到了EMIO[15:13]。AXI DMA的内部配置，地址分配等，都保持不变。

支持cache同步的MPSoC AXI DMA连接图



4.4. 硬件同步Cache的数据通道的Linux驱动设计

4.4.1. Linux Kernel选项

在cache同步的工程里，需要使用GPIO的EMIO管脚，Linux内核需要指出GPIO，请选择“GPIO_ZYNQ”。不过Xilinx Linux的缺省配置，都包含这个配置项。可以在Linux的配置文件.config搜索CONFIG_GPIO_ZYNQ，确认Linux内核包含“GPIO_ZYNQ”。

```
CONFIG_GPIO_ZYNQ=y
```

4.4.2. Linux Devicetree

现在硬件管理Cache同步，因此在Devicetree里给DMA增加Cache同步属性“dma-coherent”，使Linux跳过Cache刷新操作，节省CPU时间。

```
&axi_dma_0 {
    status = "okay";
    dma-coherent;
    };
```

4.4.2.1. 统一的Linux Devicetree

为了便于使用，在双功能的Linux Devicetree的基础上增加Cache同步属性“dma-coherent”，从而同时支持内核测试和应用程序测试。

```
/include/ "system-conf.dtsi"

/delete-node/ &axi_dma_0;

/ {
    chosen {
            bootargs = "cpuidle.off=1";
       };

    axi_dma_0: dma@a0001000 {
        #dma-cells = <1>;
        clock-names = "s_axi_lite_aclk", "m_axi_sg_aclk", "m_axi_mm2s_aclk",
```

```
"m_axi_s2mm_aclk";
        clocks = <&zynqmp_clk 71>, <&zynqmp_clk 71>, <&zynqmp_clk 71>, <&zynqmp_clk 71>;
        compatible = "xlnx,axi-dma-7.1", "xlnx,axi-dma-1.00.a";
        interrupt-names = "mm2s_introut", "s2mm_introut";
        interrupt-parent = <&gic>;
        interrupts = <0 89 4 0 90 4>;
        reg = <0x0 0xa0001000 0x0 0x1000>;
        xlnx,addrwidth = <0x28>;
        xlnx,include-sg ;
        xlnx,sg-length-width = <0x1a>;
        dma-coherent;
        status = "okay";

        dma-channel@a0001000 {
            compatible = "xlnx,axi-dma-mm2s-channel";
            dma-channels = <0x1>;
            interrupts = <0 89 4>;
            xlnx,datawidth = <0x40>;
            xlnx,device-id = <0x1>;
            xlnx,include-dre ;
        };
        dma-channel@a0001030 {
            compatible = "xlnx,axi-dma-s2mm-channel";
            dma-channels = <0x1>;
            interrupts = <0 90 4>;
            xlnx,datawidth = <0x40>;
            xlnx,device-id = <0x2>;
            xlnx,include-dre ;
        };
    };

    axidmatest_1: axidmatest@1 {
            compatible ="xlnx,axi-dma-test-1.00.a";
               dmas = <&axi_dma_0 0
                       &axi_dma_0 1>;
            dma-names = "axidma0", "axidma1";
    } ;

    axidma_chrdev: axidma_chrdev@0 {
        compatible = "xlnx,axidma-chrdev";
        dmas = <&axi_dma_0 0 &axi_dma_0 1>;
        dma-names = "tx_channel", "rx_channel";
    };
};
```

4.5. 使用GPIO设置AXI信号

测试中，使用GPIO EMIO管脚来设置AxCACHE[3:0]和AxPROT[1]。
对于GPIO的使用，具体情况请参考Linux GPIO Driver， GPIO User Space App。

4.5.1. 系统GPIO信息

Linux系统启动后，在/sys/class/gpio/创建了相关节点。在ZCU106单板上，MPSoC PS的GPIO号从338开始，其中有78个是GPIO MIO管脚，因此GPIO EMIO管脚从416开始。

```
root@xilinx-zcu106-2019_1:~# ls /sys/class/gpio/
export       gpiochip306  gpiochip322  gpiochip338  unexport
root@xilinx-zcu106-2019_1:~# ls /sys/class/gpio/ -l
export
gpiochip306 -> ../../devices/platform/amba/ff020000.i2c/i2c-0/0-
0021/gpio/gpiochip306
gpiochip322 -> ../../devices/platform/amba/ff020000.i2c/i2c-0/0-
0020/gpio/gpiochip322
gpiochip338 -> ../../devices/platform/amba/ff0a0000.gpio/gpio/gpiochip338
unexport
root@xilinx-zcu106-2019_1:~# ls /sys/class/gpio/gpiochip338/
base       device     label      ngpio      power      subsystem  uevent
root@xilinx-zcu106-2019_1:~# cat /sys/class/gpio/gpiochip338/base
338
root@xilinx-zcu106-2019_1:~# cat /sys/class/gpio/gpiochip338/label
zynqmp_gpio
root@xilinx-zcu106-2019_1:~# cat /sys/class/gpio/gpiochip338/ngpio
174
```

4.5.2. GPIO设置脚本

GPIO操作有点复杂，因此写了一个脚本，完成设置Cache相关的AXI信号的工作。

```
#/bin/bash

function gpio_output_func()
{
    # MPSoC EMIO GPIO number.
    gpio_num=$[338+78+$1]

    echo "Set GPIO number: $gpio_num to value: $2"

    if [ ! -f /sys/class/gpio/gpio$gpio_num/direction ]; then
        #  Export a GPIO pin
        echo $gpio_num > /sys/class/gpio/export
    fi

    #  Read the direction and value from the GPIO pin */
    gpio_direction=`cat /sys/class/gpio/gpio$gpio_num/direction`
    gpio_value=`cat /sys/class/gpio/gpio$gpio_num/value`
    echo "GPIO number: $gpio_num previous direction: $gpio_direction, previous
value: $gpio_value"

    #  Set the direction to an output and write a value 1 to GPIO pin */
    echo out > /sys/class/gpio/gpio$gpio_num/direction
    echo $2 > /sys/class/gpio/gpio$gpio_num/value

    gpio_direction=`cat /sys/class/gpio/gpio$gpio_num/direction`
    gpio_value=`cat /sys/class/gpio/gpio$gpio_num/value`
    echo "GPIO number: $gpio_num current direction: $gpio_direction, current value:
$gpio_value"

}

# Check GPIO chip
gpio_chip=`cat /sys/class/gpio/gpiochip338/label`
```

```
gpio_base=`cat /sys/class/gpio/gpiochip338/base`
gpio_number=`cat /sys/class/gpio/gpiochip338/ngpio`
echo "MPSoC GPIO information: chip: $gpio_chip, base: $gpio_base, number:
$gpio_number."

# https://www.xilinx.com/support/answers/69446.html
# awcache, emio[3:0], 1111, Write-back Read and Write-allocate
gpio_output_func 0 1
gpio_output_func 1 1
gpio_output_func 2 1
gpio_output_func 3 1

# awprot, emio[7:5]
# AxPROT[1] needs to match the security setting and processor exception level for
the memory region.
# Bare metal application, EL3, secure memory accesses. AxPROT[1] : 0
# Linux kernel/application, EL1/EL0, non-secure memory accesses. AxPROT[1] : 1
gpio_output_func 5 0
gpio_output_func 6 1
gpio_output_func 7 0

# arcache, emio[11:8], 1111, Write-back Read and Write-allocate
gpio_output_func 8 1
gpio_output_func 9 1
gpio_output_func 10 1
gpio_output_func 11 1

# arprot, emio[15:13]
# Linux kernel/application, EL1/EL0, non-secure memory accesses. AxPROT[1] : 1
gpio_output_func 13 0
gpio_output_func 14 1
gpio_output_func 15 0
```

4.5.3. GPIO设置记录

使用前述脚本，设置GPIO很方便。GPIO设置记录如下：

```
root@xilinx-zcu106-2019_1:~# ls -l
total 896
-rwxr-x---    1 root     root           944 Nov  1 09:06 axi_dma_test_top.sh
-rwxr-x---    1 root     root           173 Nov  1 09:06 cp_dma_modules.sh
-rwxr-x---    1 root     root          2705 Nov  1 09:06 gpio_set_hpc_cache.sh
-rwxr-x---    1 root     root          1991 Nov  1 09:06 gpio_single_output.sh

root@xilinx-zcu106-2019_1:~# ./gpio_set_hpc_cache.sh
MPSoC GPIO information: chip: zynqmp_gpio, base: 338, number: 174.
Set GPIO number: 416 to value: 1
GPIO number: 416 previous direction: in, previous value: 0
GPIO number: 416 current direction: out, current value: 1
Set GPIO number: 417 to value: 1
GPIO number: 417 previous direction: in, previous value: 0
GPIO number: 417 current direction: out, current value: 1
Set GPIO number: 418 to value: 1
GPIO number: 418 previous direction: in, previous value: 0
GPIO number: 418 current direction: out, current value: 1
Set GPIO number: 419 to value: 1
GPIO number: 419 previous direction: in, previous value: 0
```

```
GPIO number: 419 current direction: out, current value: 1
Set GPIO number: 421 to value: 0
GPIO number: 421 previous direction: in, previous value: 0
GPIO number: 421 current direction: out, current value: 0
Set GPIO number: 422 to value: 1
GPIO number: 422 previous direction: in, previous value: 0
GPIO number: 422 current direction: out, current value: 1
Set GPIO number: 423 to value: 0
GPIO number: 423 previous direction: in, previous value: 0
GPIO number: 423 current direction: out, current value: 0
Set GPIO number: 424 to value: 1
GPIO number: 424 previous direction: in, previous value: 0
GPIO number: 424 current direction: out, current value: 1
Set GPIO number: 425 to value: 1
GPIO number: 425 previous direction: in, previous value: 0
GPIO number: 425 current direction: out, current value: 1
Set GPIO number: 426 to value: 1
GPIO number: 426 previous direction: in, previous value: 0
GPIO number: 426 current direction: out, current value: 1
Set GPIO number: 427 to value: 1
GPIO number: 427 previous direction: in, previous value: 0
GPIO number: 427 current direction: out, current value: 1
Set GPIO number: 429 to value: 0
GPIO number: 429 previous direction: in, previous value: 0
GPIO number: 429 current direction: out, current value: 0
Set GPIO number: 430 to value: 1
GPIO number: 430 previous direction: in, previous value: 0
GPIO number: 430 current direction: out, current value: 1
Set GPIO number: 431 to value: 0
GPIO number: 431 previous direction: in, previous value: 0
GPIO number: 431 current direction: out, current value: 0
```

4.6. 硬件同步Cache的数据通道的Linux内核测试

如果不正确设置cache相关的AXI信号，运行AXI DMA内核模块测试，会出现发送超时错误，错误代码10。

```
root@xilinx-zcu106-2019_1:~# insmod axidmatest.ko
[  111.124426] dmatest: Started 1 threads using dma1chan0 dma1chan1
[  112.125900] xilinx-vdma a0001000.dma: Cannot start channel (____ptrval____):
10009
[  112.133478] xilinx-vdma a0001000.dma: Channel (____ptrval____) has errors 10, cdr
15aa0000 tdr 15aa0500

[  142.315600] dma1chan0-dma1c: #0: tx test timed out
[  143.321632] xilinx-vdma a0001000.dma: Cannot start channel (____ptrval____):
20009
[  143.329210] xilinx-vdma a0001000.dma: Channel (____ptrval____) has errors 10, cdr
15aa0580 tdr 15aa0a80

[  175.083600] dma1chan0-dma1c: #1: tx test timed out
[  176.089632] xilinx-vdma a0001000.dma: Cannot start channel (____ptrval____):
30009
[  176.097208] xilinx-vdma a0001000.dma: Channel (____ptrval____) has errors 10, cdr
15aa0b00 tdr 15aa1000
```

```
[  207.851608] dma1chan0-dma1c: #2: tx test timed out
[  208.857699] xilinx-vdma a0001000.dma: Cannot start channel (____ptrval____):
40009
[  208.865271] xilinx-vdma a0001000.dma: Channel (____ptrval____) has errors 10, cdr
15aa1080 tdr 15aa1580

[  240.619602] dma1chan0-dma1c: #3: tx test timed out
[  241.625660] xilinx-vdma a0001000.dma: Cannot start channel (____ptrval____):
50009
[  241.633238] xilinx-vdma a0001000.dma: Channel (____ptrval____) has errors 10, cdr
15aa1600 tdr 15aa1b00

[  273.387603] dma1chan0-dma1c: #4: tx test timed out
[  273.392400] dma1chan0-dma1c: terminating after 5 tests, 5 failures (status 0)
```

如果出现上述错误，请检查FSBL修改了寄存器Snoop_Control_Register_S3(0xFD6E4000)，有相关打印；Boot.bin含义初始化寄存器lpd_apu（0xFF41A040)的内容；通过GPIO设置了cache相关的AXI信号。能在Linux下读到寄存器lpd_apu（0xFF41A040)的内容，但是不能读寄存器Snoop_Control_Register_S3，会收到Bus error。

```
root@xilinx-zcu106-2019_1:~# devmem 0xFF41A040
0x00000003
root@xilinx-zcu106-2019_1:~# devmem 0xFD6E4000
Bus error
```

注意，出现测试失败的情况后，需要重启动单板。在测试中发现，如果不重启动单板，即使设置了cache相关的AXI信号，后续测试还是会失败。

如果所有设置正确，运行AXI DMA内核模块测试，没有任何错误信息，和传统数据通路的Linux内核测试的输出信息一样。

回到目录前 ***** 回到目录后

4.7. 硬件同步Cache的数据通道的应用程序测试

如果不正确设置cache相关的AXI信号，运行AXI DMA内核模块测试，会出现错误。

```
root@xilinx-zcu106-2019_1:~# ./axidma_benchmark
AXI DMA Benchmark Parameters:
        Transmit Buffer Size: 7.91 MiB
        Receive Buffer Size: 7.91 MiB
        Number of DMA Transfers: 1000 transfers

[  140.621473] axidma axidma: DMA mask not set

Using transmit channel 1 and receive channel 2.

[  140.716763] xilinx-vdma a0001000.dma: Channel (____ptrval____) has errors 10, cdr
15aa0000 tdr 15aa0000
[  141.716756] xilinx-vdma a0001000.dma: Cannot start channel (____ptrval____):
10009
[  151.783898] axidma: axidma_dma.c: axidma_start_transfer: 301: DMA receive
transaction timed out.
```

```
Failed to perform the AXI DMA read-write transfer: Timer expired
```

注意，出现测试失败后，也需要重启动单板。

如果正确设置cache相关的AXI信号，运行AXI DMA内核模块测试，没有任何错误信息，和传统数据通路的应用程序测试的输出信息一样。

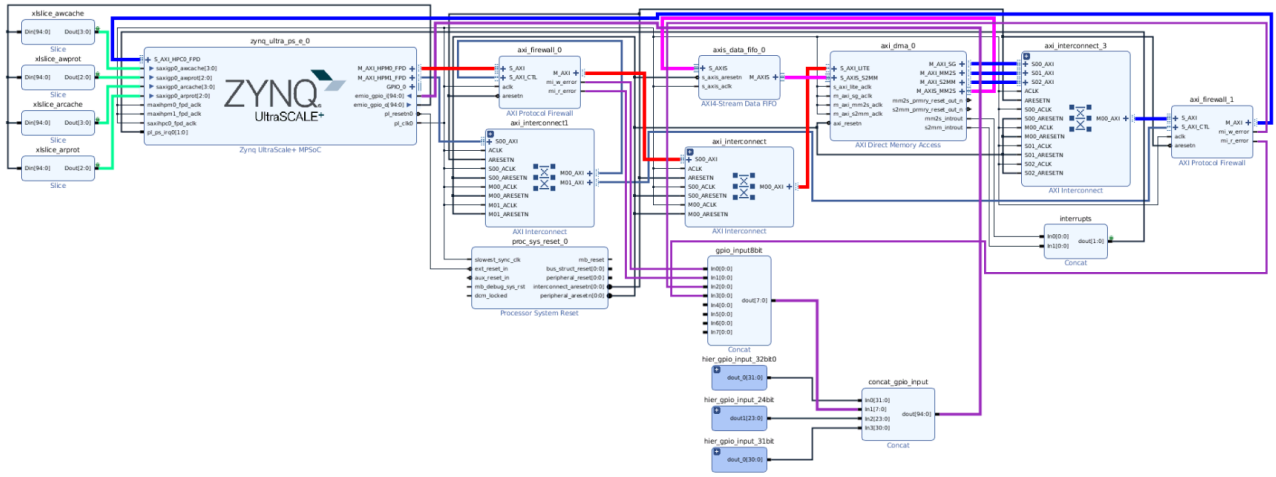
回到目录前 ***** 回到目录后

5. 数据通道的改进

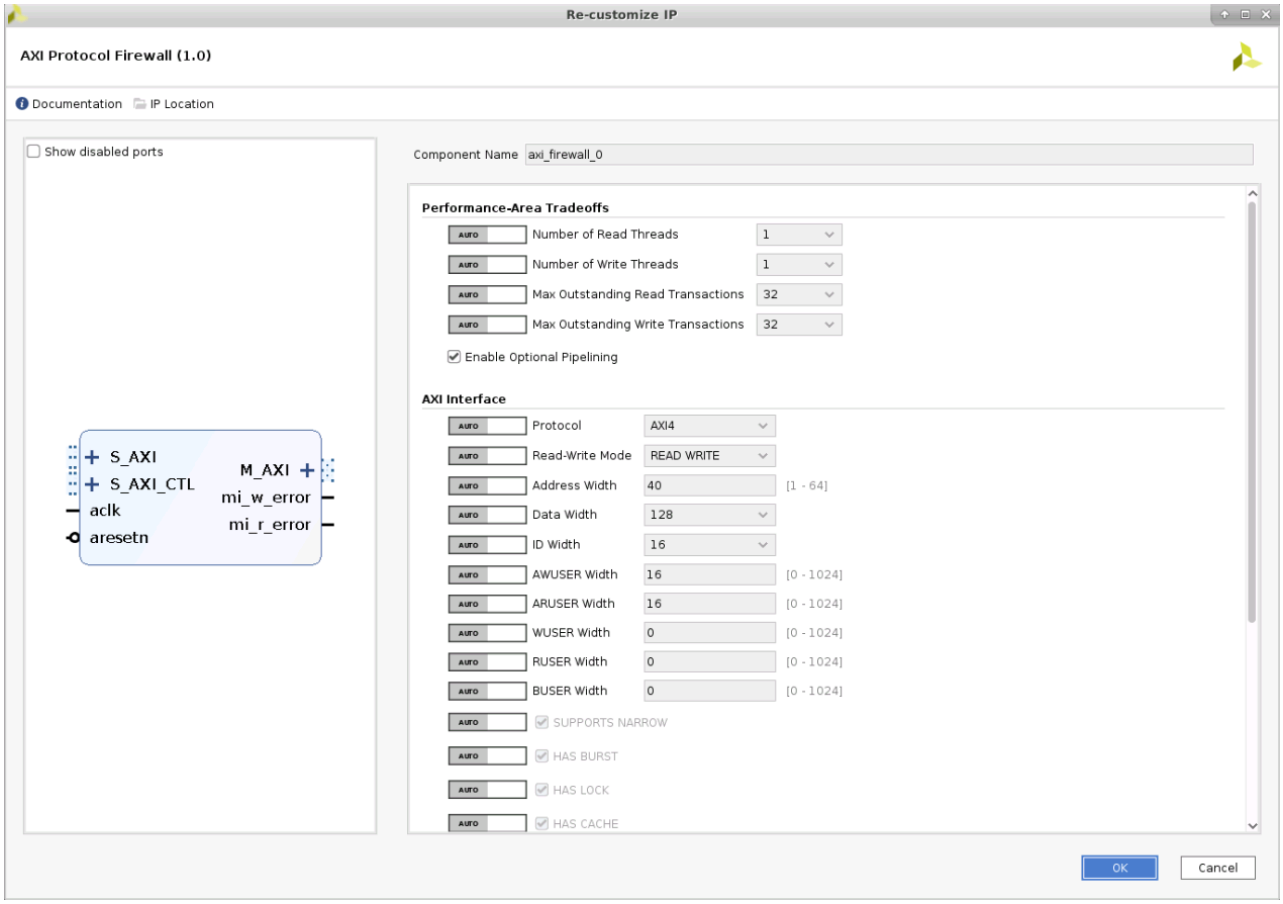
5.1. 增加AXI Firewall的硬件设计

在设计和调试过程中，可能出现意料之外的错误。如果CPU访问的AXI通道出现问题，会导致系统死机。为了帮助调试，Xilinx提供了辅助的IP，包括AXI Firewall,AXI protocol checker。AXI Firewall在AXI 波形异常时，主动结束AXI传输，保护系统不死机，同时给出出错信号。建议把AXI Firewall加在PS的输入和输出端口，用GPIO EMIO的输入端口连接AXI Firewall的错误指示信号。

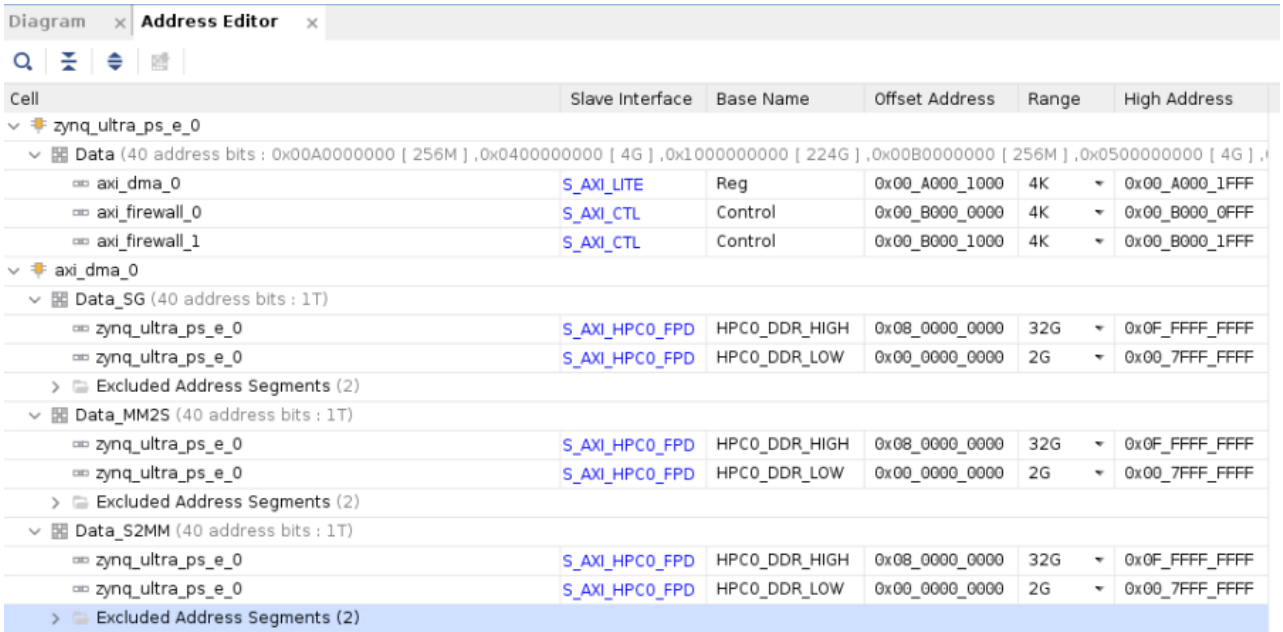
增加AXI Firewall的硬件设计连接图



AXI Firewall的参数配置



增加AXI Firewall的硬件地址分配

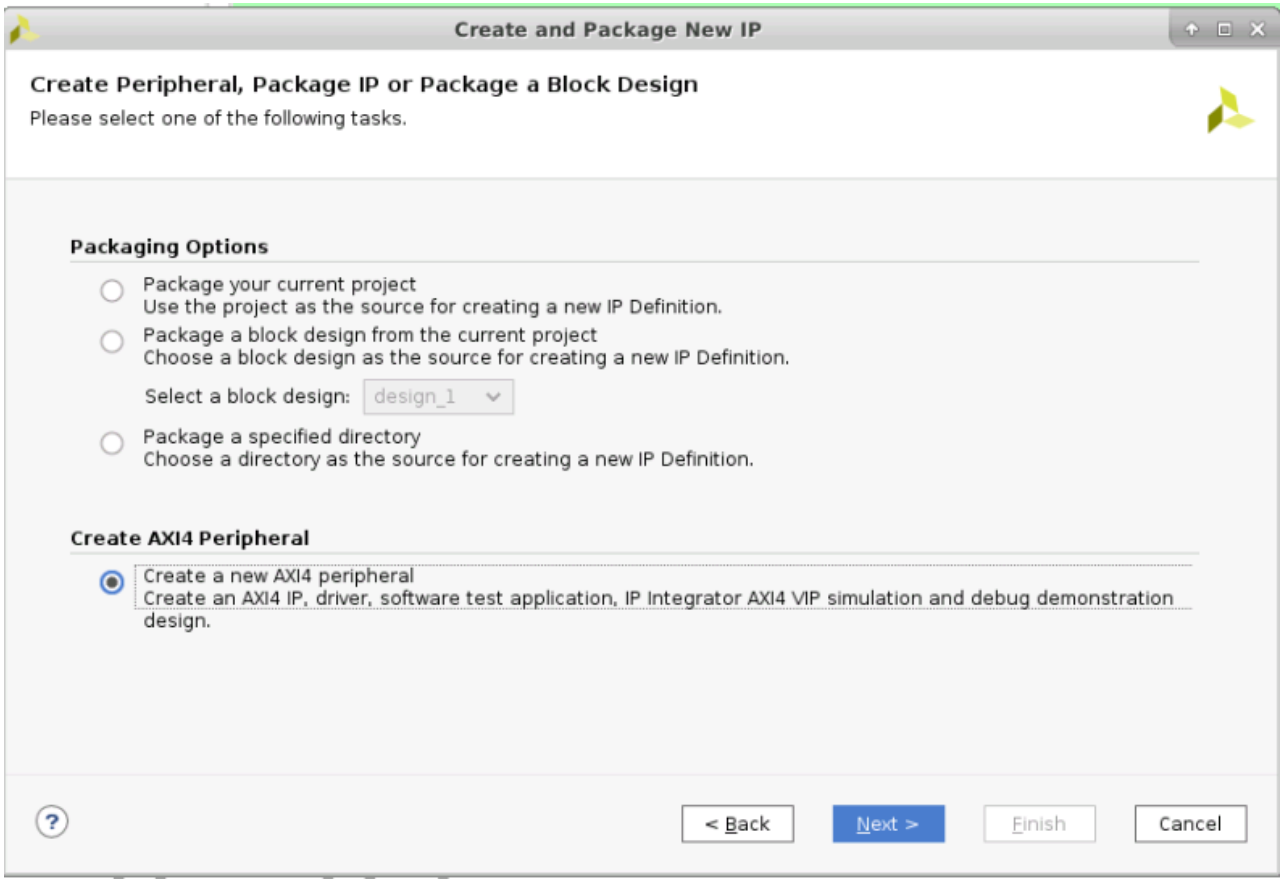


Cell	Slave Interface	Base Name	Offset Address	Range	High Address
zynq_ultra_ps_e_0					
Data (40 address bits : 0x00A0000000 [256M] ,0x0400000000 [4G] ,0x1000000000 [224G] ,0x00B0000000 [256M] ,0x0500000000 [4G] ,					
axi_dma_0	S_AXI_LITE	Reg	0x00_A000_1000	4K	0x00_A000_1FFF
axi_firewall_0	S_AXI_CTL	Control	0x00_B000_0000	4K	0x00_B000_0FFF
axi_firewall_1	S_AXI_CTL	Control	0x00_B000_1000	4K	0x00_B000_1FFF
axi_dma_0					
Data_SG (40 address bits : 1T)					
zynq_ultra_ps_e_0	S_AXI_HPC0_FPD	HPC0_DDR_HIGH	0x08_0000_0000	32G	0x0F_FFFF_FFFF
zynq_ultra_ps_e_0	S_AXI_HPC0_FPD	HPC0_DDR_LOW	0x00_0000_0000	2G	0x00_7FFF_FFFF
Excluded Address Segments (2)					
Data_MM2S (40 address bits : 1T)					
zynq_ultra_ps_e_0	S_AXI_HPC0_FPD	HPC0_DDR_HIGH	0x08_0000_0000	32G	0x0F_FFFF_FFFF
zynq_ultra_ps_e_0	S_AXI_HPC0_FPD	HPC0_DDR_LOW	0x00_0000_0000	2G	0x00_7FFF_FFFF
Excluded Address Segments (2)					
Data_S2MM (40 address bits : 1T)					
zynq_ultra_ps_e_0	S_AXI_HPC0_FPD	HPC0_DDR_HIGH	0x08_0000_0000	32G	0x0F_FFFF_FFFF
zynq_ultra_ps_e_0	S_AXI_HPC0_FPD	HPC0_DDR_LOW	0x00_0000_0000	2G	0x00_7FFF_FFFF
Excluded Address Segments (2)					

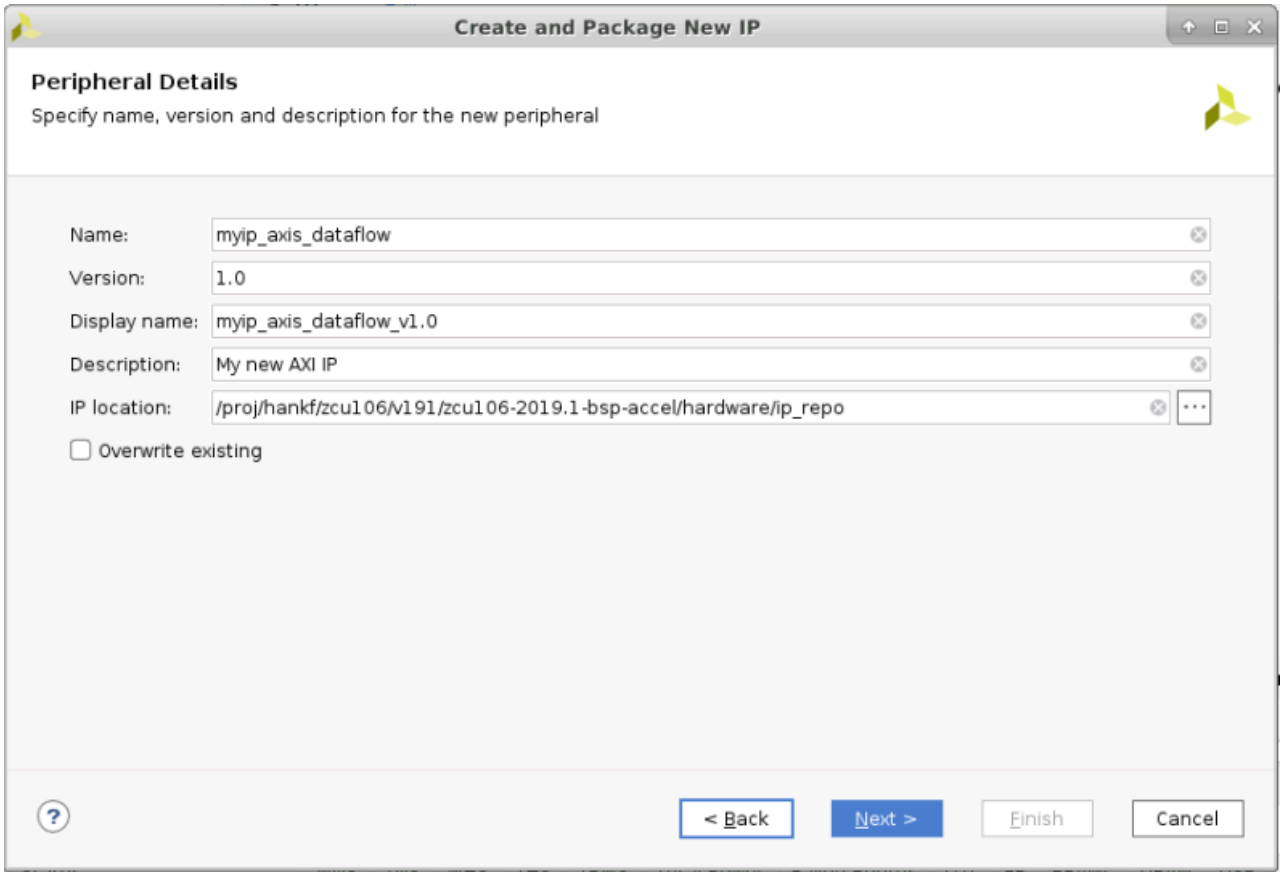
5.2. AXI stream设备示例代码

Vivado提供AXI的示例代码。在Vivado的菜单“Tools”下，选择“Create and Package New IP...”, 即可定制AXI stream设备示例代码。

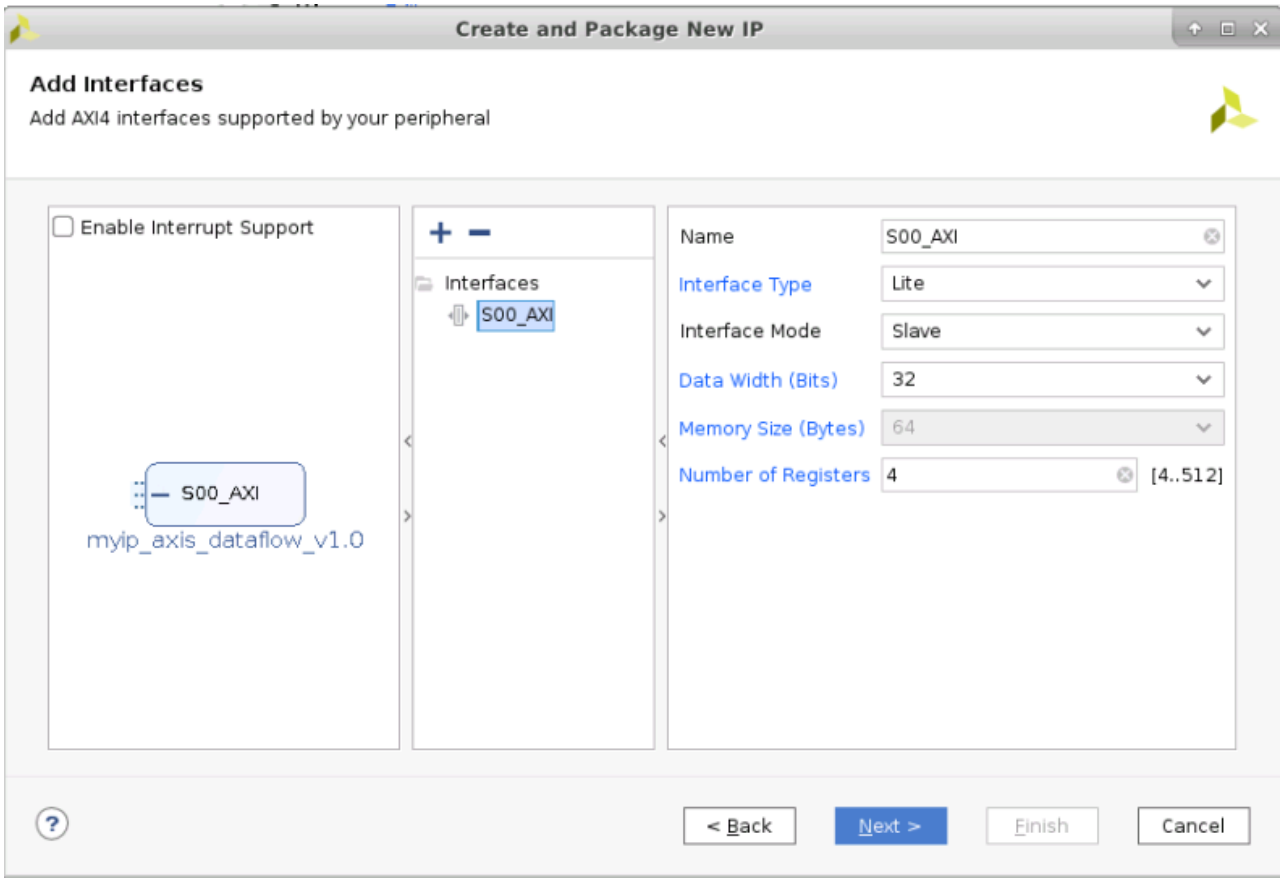
AXI stream设备示例图，步骤1



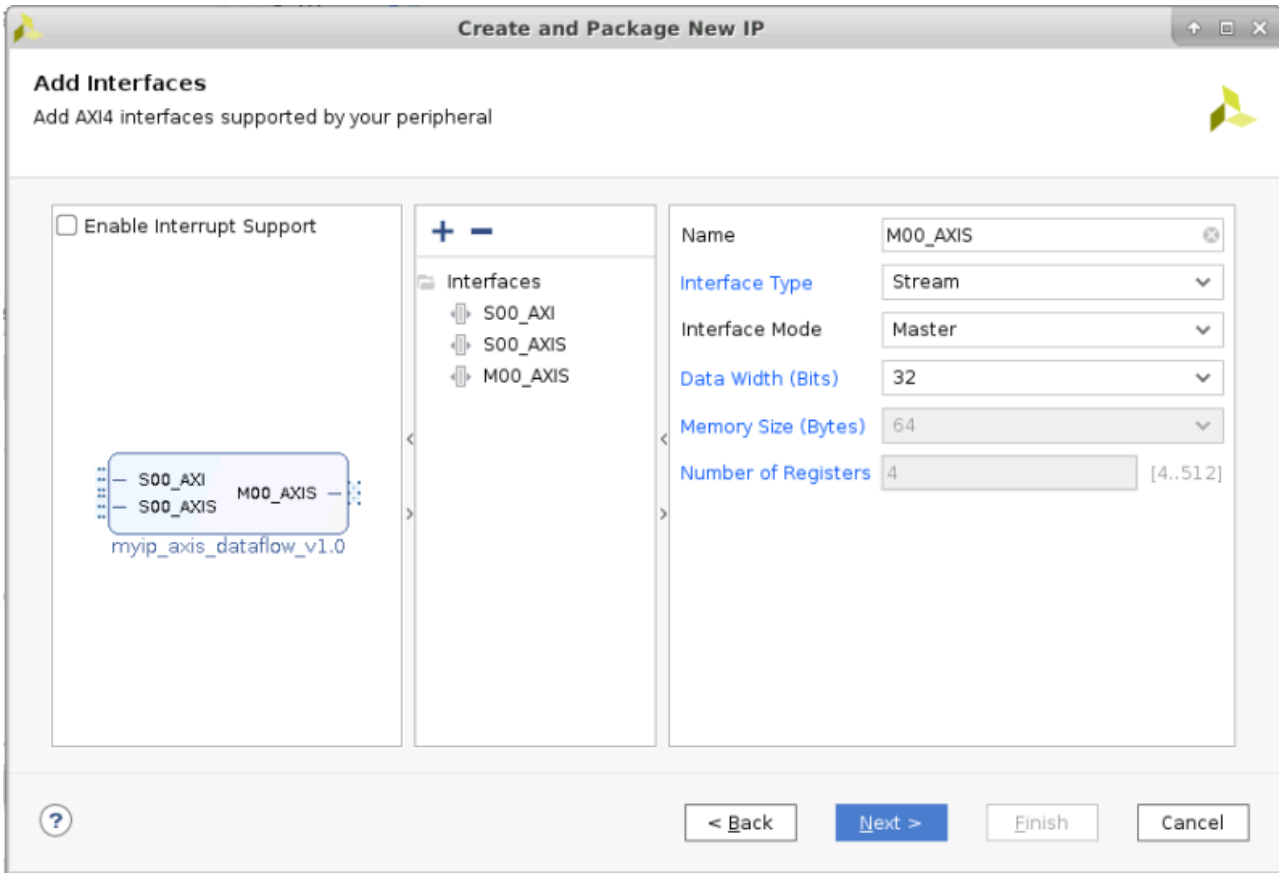
AXI stream设备示例图，步骤2



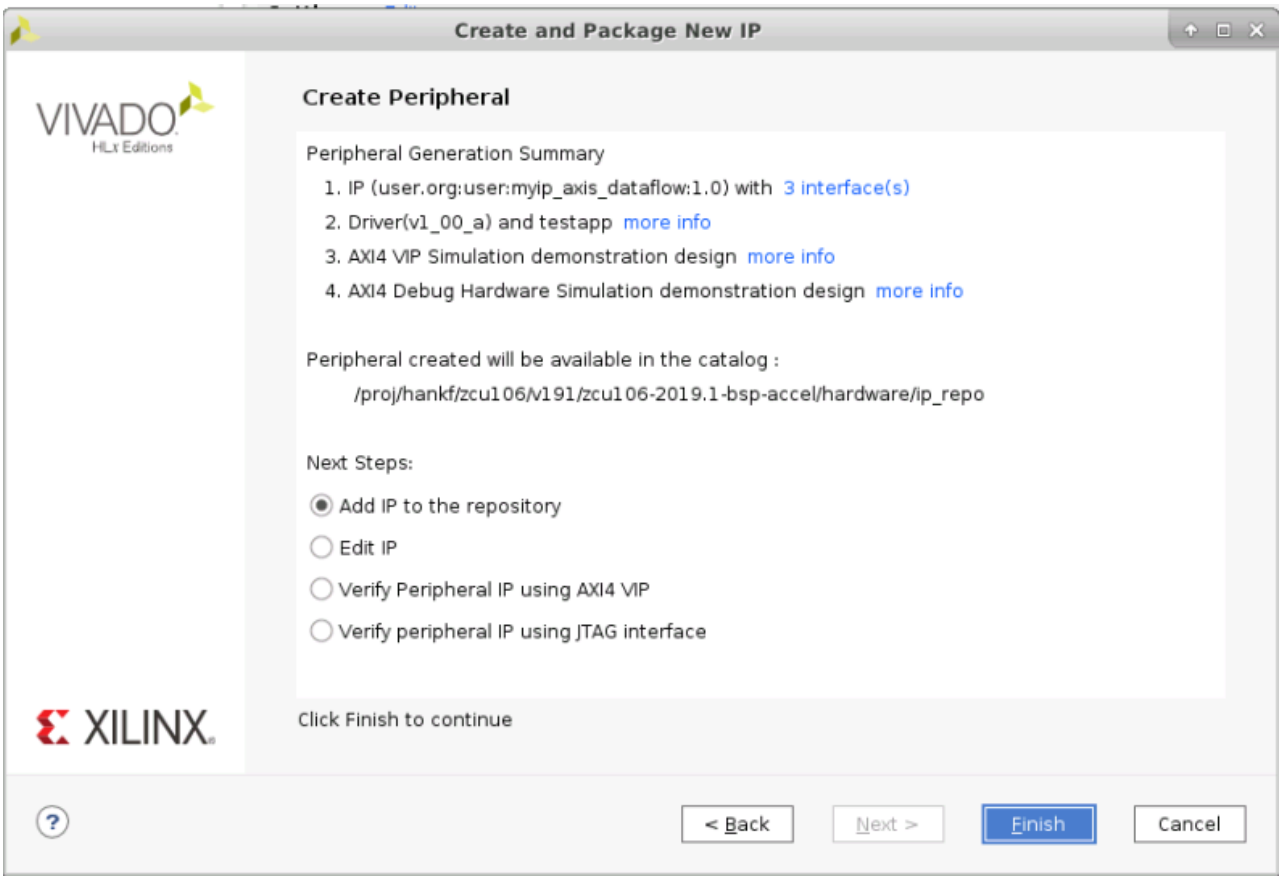
AXI stream设备示例图，步骤3



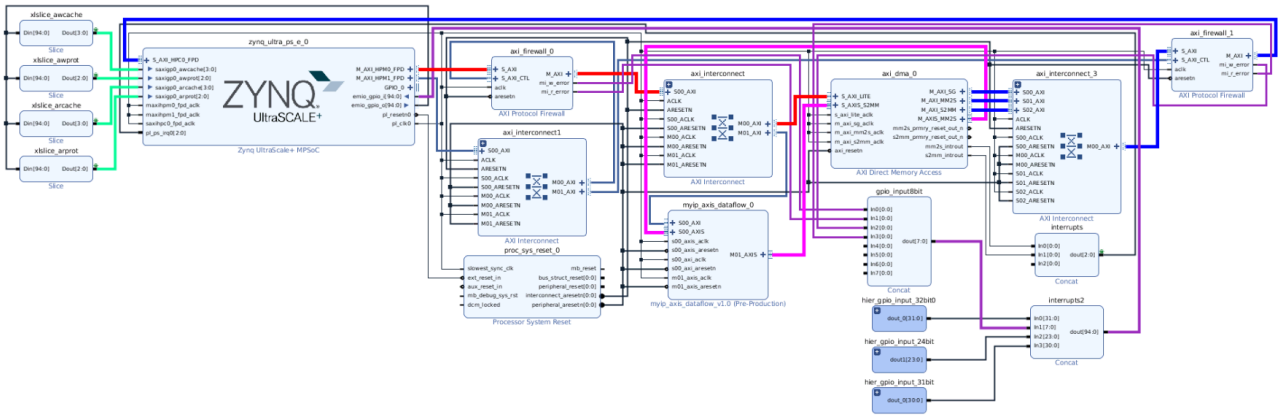
AXI stream设备示例图，步骤4



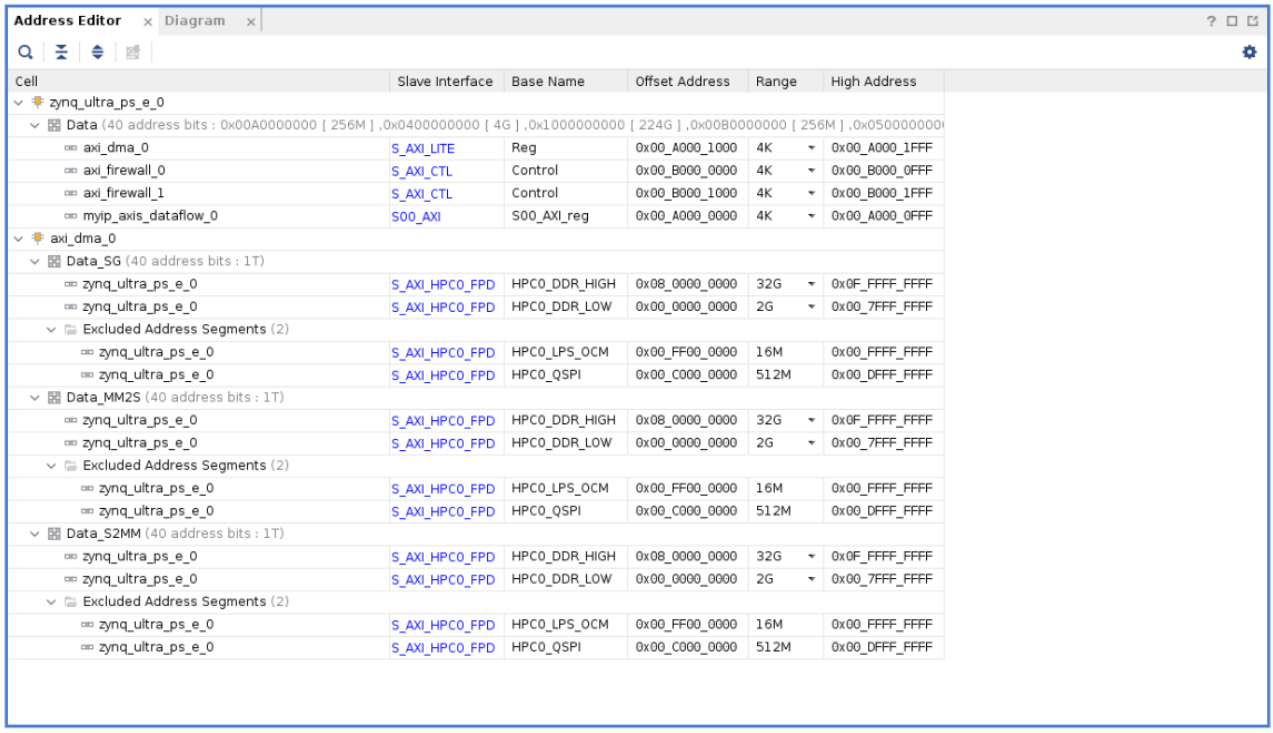
AXI stream设备示例图，步骤5



使用AXI stream设备示例的硬件设计连接图



使用AXI stream设备示例的硬件地址分配图



Address Editor | Diagram

Cell	Slave Interface	Base Name	Offset Address	Range	High Address
zynq_ultra_ps_e_0					
Data (40 address bits : 0x00A0000000 [256M] ,0x0400000000 [4G] ,0x1000000000 [224G] ,0x00B0000000 [256M] ,0x050000000					
axi_dma_0	S_AXI_LITE	Reg	0x00_A000_1000	4K	0x00_A000_1FFF
axi_firewall_0	S_AXI_CTL	Control	0x00_B000_0000	4K	0x00_B000_0FFF
axi_firewall_1	S_AXI_CTL	Control	0x00_B000_1000	4K	0x00_B000_1FFF
myip_axis_dataflow_0	S00_AXI	S00_AXI_reg	0x00_A000_0000	4K	0x00_A000_0FFF
axi_dma_0					
Data_SG (40 address bits : 1T)					
zynq_ultra_ps_e_0	S_AXI_HPC0_FPD	HPC0_DDR_HIGH	0x08_0000_0000	32G	0x0F_FFFF_FFFF
zynq_ultra_ps_e_0	S_AXI_HPC0_FPD	HPC0_DDR_LOW	0x00_0000_0000	2G	0x00_7FFF_FFFF
Excluded Address Segments (2)					
zynq_ultra_ps_e_0	S_AXI_HPC0_FPD	HPC0_LPS_OCM	0x00_FF00_0000	16M	0x00_FFFF_FFFF
zynq_ultra_ps_e_0	S_AXI_HPC0_FPD	HPC0_QSPI	0x00_C000_0000	512M	0x00_DFFF_FFFF
Data_MM2S (40 address bits : 1T)					
zynq_ultra_ps_e_0	S_AXI_HPC0_FPD	HPC0_DDR_HIGH	0x08_0000_0000	32G	0x0F_FFFF_FFFF
zynq_ultra_ps_e_0	S_AXI_HPC0_FPD	HPC0_DDR_LOW	0x00_0000_0000	2G	0x00_7FFF_FFFF
Excluded Address Segments (2)					
zynq_ultra_ps_e_0	S_AXI_HPC0_FPD	HPC0_LPS_OCM	0x00_FF00_0000	16M	0x00_FFFF_FFFF
zynq_ultra_ps_e_0	S_AXI_HPC0_FPD	HPC0_QSPI	0x00_C000_0000	512M	0x00_DFFF_FFFF
Data_S2MM (40 address bits : 1T)					
zynq_ultra_ps_e_0	S_AXI_HPC0_FPD	HPC0_DDR_HIGH	0x08_0000_0000	32G	0x0F_FFFF_FFFF
zynq_ultra_ps_e_0	S_AXI_HPC0_FPD	HPC0_DDR_LOW	0x00_0000_0000	2G	0x00_7FFF_FFFF
Excluded Address Segments (2)					
zynq_ultra_ps_e_0	S_AXI_HPC0_FPD	HPC0_LPS_OCM	0x00_FF00_0000	16M	0x00_FFFF_FFFF
zynq_ultra_ps_e_0	S_AXI_HPC0_FPD	HPC0_QSPI	0x00_C000_0000	512M	0x00_DFFF_FFFF

上述例子，也能正常运行AXI DMA的内核测试和应用程序测试。

5.3. AXI DMA 性能

5.3.1. 提升频率

上述测试工程，AXI DMA的运行频率是100MHz。提升频率到250Mhz，发现性能和之前一样。

5.3.2. 增加AXI端口

上述测试工程，只使用了一个PS的AXI端口。更改设计，提升频率到250Mhz，使mm2s和s2mm各用一个AXI端口，发现性能和之前一样。

回到目录前 ***** 回到目录后

6. 设计文件

为了便于客户移植，把所有设计相关文件打包，封装在文件zcu106-2019.1-bsp-accel-1113-1730-axi-dma-firewall-hw-coherent.bsp里。

6.1. 硬件工程

PetaLinux BSP文件里，提供了HDF文件project-spec\hw-descriptionsystem.hdf，PetaLinux工程的配置，devicetree文件等。
HDF文件project-spec\hw-description\system.hdf，是一个压缩文件，得到tcl文件design_1_bd.tcl,可以在Vivado里执行,恢复硬件工程。
为了便于版本管理，只提供了硬件同步Cache、带AXI Firewall的硬件工程。如果不需要这些特性，可以删除设置AXI 信号的GPIO信号，AXI Firewall。

6.2. Linux 映象

目录pre-built\linux\images下含有BOOT.BIN，image.ub，可以直接运行测试。

6.3. 启动文件

目录pre-built\linux\images下还有xfsbl_main.c，regs.init，bootgen.bif，用于恢复FSBL，BOOT.BIN。

6.4. 应用层测试程序

目录re-built\linux\bperez77_xilinx_axidma-master含有应用层测试程序的代码。
目录pre-built\linux\bperez77_xilinx_axidma-master\outputs 含有已经编译好的应用层测试的驱动和程序。

6.5. 测试辅助脚本

目录/pre-built/linux/scripts下含有几个测试脚本，用于简化测试操作。其中axi_dma_test_prep.sh可以把所有文件从SD卡，拷贝到当前目录。axi_dma_test_run_kernel_one.sh运行一次内核态的测试。
axi_dma_test_run_user_benchmark.sh运行一次用户态的测试。

```
1. gpio_set_hpc_cache.sh
2. gpio_single_output.sh
3. axi_dma_test_prep.sh
4. axi_dma_test_run_kernel_big.sh
5. axi_dma_test_run_kernel_one.sh
6. axi_dma_test_run_user_benchmark.sh
7. cp_dma_modules.sh
8. gpio_set_hpc_cache.sh
9. gpio_single_output.sh
```

6.6. 测试运行记录

为了方便理解，在目录pre-built\linux\images下提供了完整的Linux运行记录， linux-axi-dma-test-log.txt。

回到目录前 ***** 回到目录后

7. 参考文档

1. UG1085 Zynq UltraScale+ Device Technical Reference Manual v2.1 August 21, 2019
2. UG1228 Zynq UltraScale+ MPSoC Embedded Design Methodology Guide v1.0 March 31, 2017
3. UG1137 Zynq UltraScale+ MPSoC Software Developer Guide v10.0 June 26, 2019
4. UG1209 Zynq UltraScale+ MPSoC: Embedded Design Tutorial v2018.2 July 31, 2018
5. UG1037 AXI Reference Guide v4.0 July 15, 2017
6. PG021 AXI DMA v7.1 LogiCORE IP Product Guide v7.1 June 14, 2019
7. PB042 LogiCORE IP Slice Product Brief v1.0 April 6, 2016
8. PG081 AXI4-Stream Accelerator Adapter v2.1 November 18, 2015
9. ZCU106 Evaluation Board User Guide UG1244 v1.0 March 28, 2018
10. AMBA AXI and ACE Protocol Specification, ARM IHI 0022E
11. Zynq UltraScale MPSoC Cache Coherency
12. AR# 69446 Zynq UltraScale+ MPSoC Example Design - Use AXI HPC port to perform coherent transfers
13. Xilinx SoftIP DMA'S Linux driver.
14. Linux GPIO Driver
15. GPIO User Space App
16. Connecting XSDB to Linux CPU idle
17. Brandon Perez Xilinx AXI DMA Driver and Library
18. Kawazome Ichiro User space mappable DMA Buffer
19. Linux DMA From User Space
20. A Domain-Specific Architecture for Deep Neural Networks
21. Devicetree Specification Release v0.2 Devicetree.org 20 December 2017
22. FINFET AND FD SOI: MARKET AND COST ANALYSIS
23. Device Tree（二）：基本概念

回到目录前 ***** 回到目录后

付汉杰 hankf@amd.com

标签: cache , coherent , dma , mpsoc

« 上一篇： [分享]升级MPSoC Linux LTS 版本和Realtime版本
» 下一篇： [分享] SDK 2018.3烧写没有DDR的单板的Flash

posted @ 2019-11-28 15:33 HankFu 阅读(4471) 评论(1) 编辑 收藏 举报