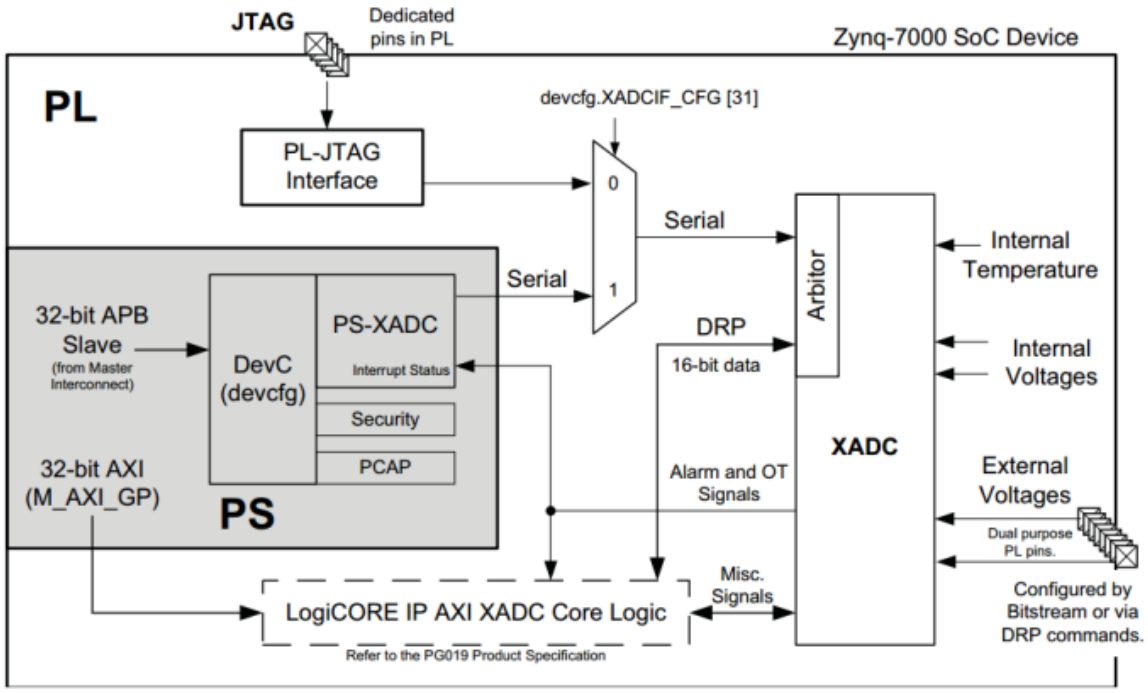
[L29]PS-PL常用通信简述(XADC,BRAM,DMA)

PS-PL通信有很多种,比如通过SDK,通常调用函数XXX_LookupConfig然后XXX_CfgInitizlize能完成,而且SDK会提供例子,所以这里不会说太多,这里主要讨论Linux下的通信.

XADC有多条通信路径,可以经过PL,也可以通过串行复用器,所以有两个访问地址.



参考配置(注意他们在不同的子节点上!):

```
xadc@f8007100 {
    compatible = "xlnx,zynq-xadc-1.00.a"; // 从串行复用器访问,在amba上.
    reg = <0xf8007100 0x20>;
    interrupts = <0 7 4>;
    interrupt-parent = <&gic>; // 替换成实际的中断控制器,比如&intc
    clocks = <&pcap_clk>; // 替换成实际时钟,比如&clkc 12

    xlnx,channels {
        #address-cells = <1>;
```

```
        #size-cells = <0>;
        channel@0 {
            reg = <0>;
        };
        channel@1 {
            reg = <1>;
        };
        channel@8 {
            reg = <8>;
        };
    };
};

xadc@43200000 {
    compatible = "xlnx,axi-xadc-1.00.a"; // 从PL侧访问,在amba_pl上.
    reg = <0x43200000 0x1000>;
    interrupts = <0 53 4>;
    interrupt-parent = <&gic>; // 替换成实际的中断控制器,比如&intc
    clocks = <&fpga1_clk>; // 替换成实际时钟,比如&clkc 15

    xlnx,channels {
        #address-cells = <1>;
        #size-cells = <0>;
        channel@0 {
            reg = <0>;
            xlnx,bipolar; // 差分模式
        };
    };
};
```

不过zynq-7000.dtsi已经给了基础配置,因此我们可以引用.

```
&adc {
    xlnx,channels {
        #address-cells = <1>;
        #size-cells = <0>;
        channel@0 {
            reg = <0>;
        };
        channel@1 {
            reg = <1>;
        };
        channel@8 {
            reg = <8>;
        };
    };
};
```

在对应的地址就能访问到:

```
#!/bin/bash
raw=`cat /sys/bus/iio/devices/iio:device0/in_temp0_raw`
offset=`cat /sys/bus/iio/devices/iio:device0/in_temp0_offset`
scale=`cat /sys/bus/iio/devices/iio:device0/in_temp0_scale`

c_temp=`echo "scale=1;(($raw + $offset) * $scale) / 1000" | bc`
f_temp=`echo "scale=1;(($c_temp * 9) / 5) + 32" | bc`

echo
echo "Zynq Temp: $c_temp C / $f_temp F"
echo
```

测试:

```
root@petalinux:~# bash get_temp.sh

Zynq Temp: 64.3 C / 147.7 F

root@petalinux:~#
```

实际Zynq的热量还是很高的,所以有可能还是要加强散热.

BRAM常用于PS-PL高速数据交换,对于PL来说,BRAM操作简单,对于PS来说,和正常的内存没什么两样,吞吐量完全看总线的意思,PORTB可以用来接PL端.

由于现在暂时只做PL端实验,所以只链接了一侧,这里比较特别的是,使用Address Editor的Range代表深度.

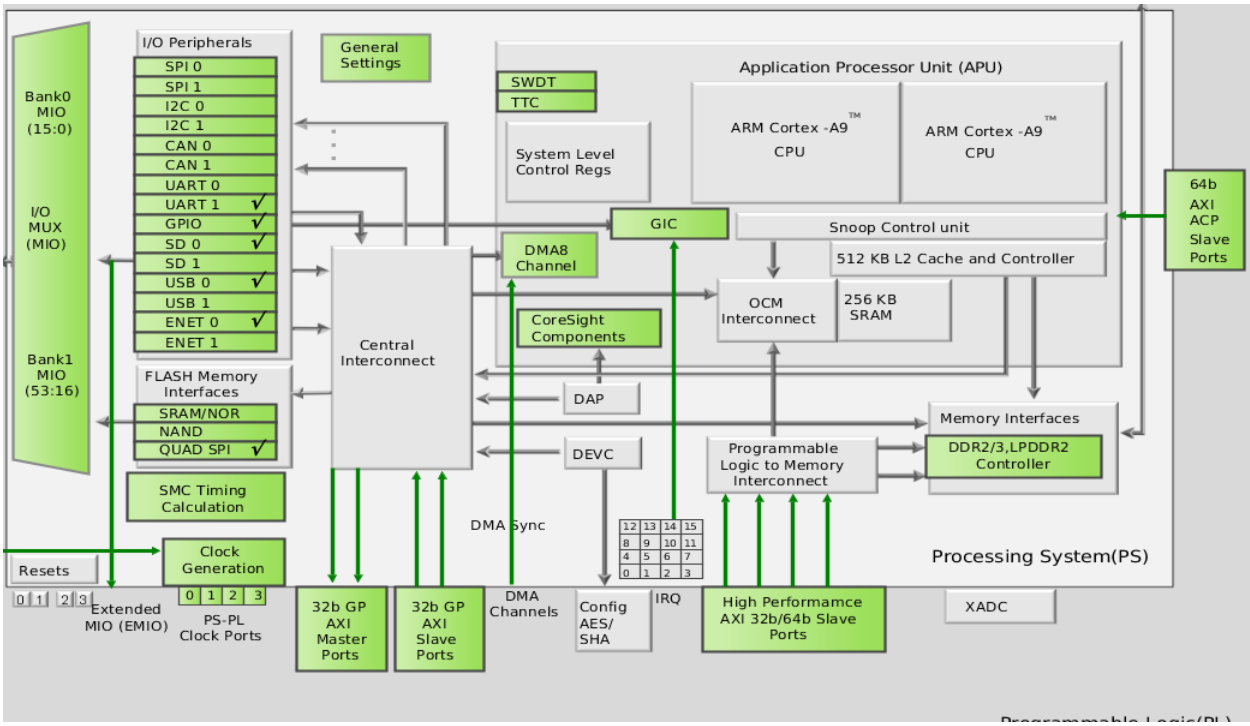
Diagram | Address Editor

Cell	Slave Interface	Base Name	Offset Address	Range	High Address
processing_system7_0					
Data (32 address bits : 0x40000000 [1G])					
axi_vdma_0	S_AXI_LITE	Reg	0x4300_0000	64K	0x4300_FFFF
v_tc_0	ctrl	Reg	0x43C0_0000	64K	0x43C0_FFFF
axi_dynclk_0	S_AXI_LITE	S_AXI_LITE_reg	0x43C1_0000	64K	0x43C1_FFFF
axi_gpio_0	S_AXI	Reg	0x4120_0000	64K	0x4120_FFFF
axi_bram_ctrl_0	S_AXI	Mem0	0x4000_0000	8K	0x4000_1FFF
axi_vdma_0					
Data_MM2S (32 address bits : 4G)					
processing_system7_0	S_AXI_HP0	HP0_DDR_LOWOCM	0x0000_0000	512M	0x1FFF_FFFF

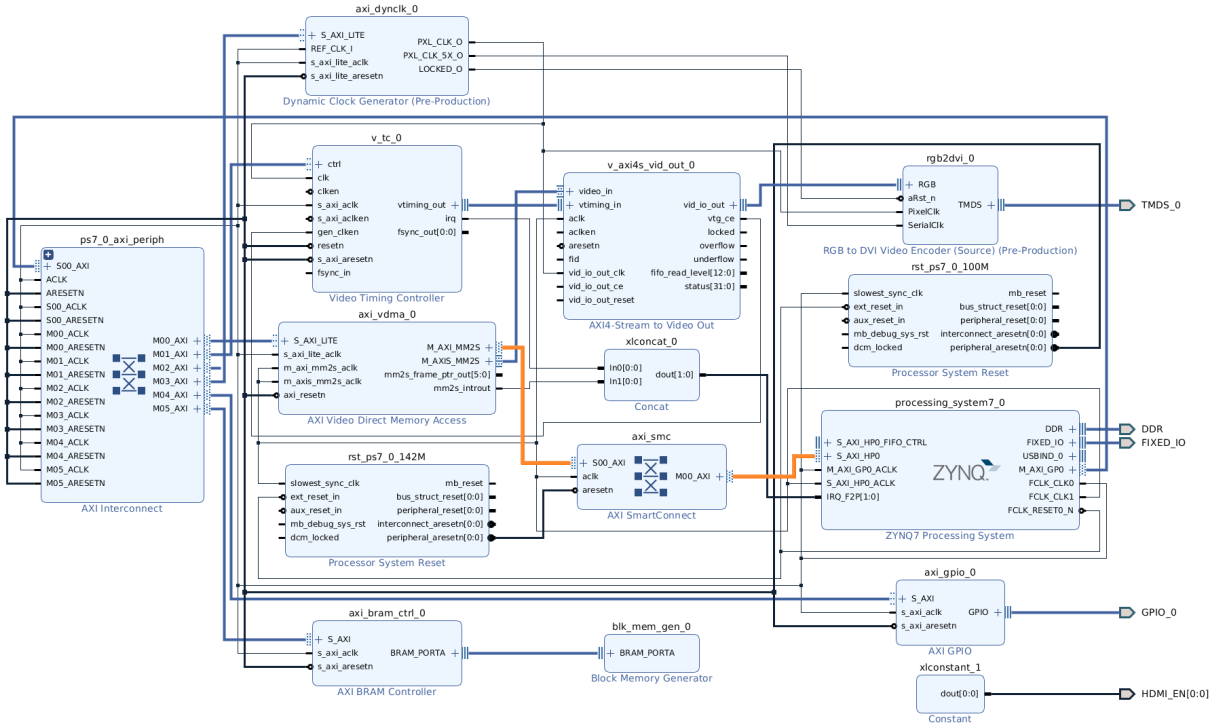
生成后通过devmem命令,mmap命令等可以直接访问它.

```
root@openamp:~# devmem 0x40000000 32
0x000000060
root@openamp:~# devmem 0x40000000 32 0x0007EF06
root@openamp:~# devmem 0x40000000 32
0x0007EF06
root@openamp:~#
```

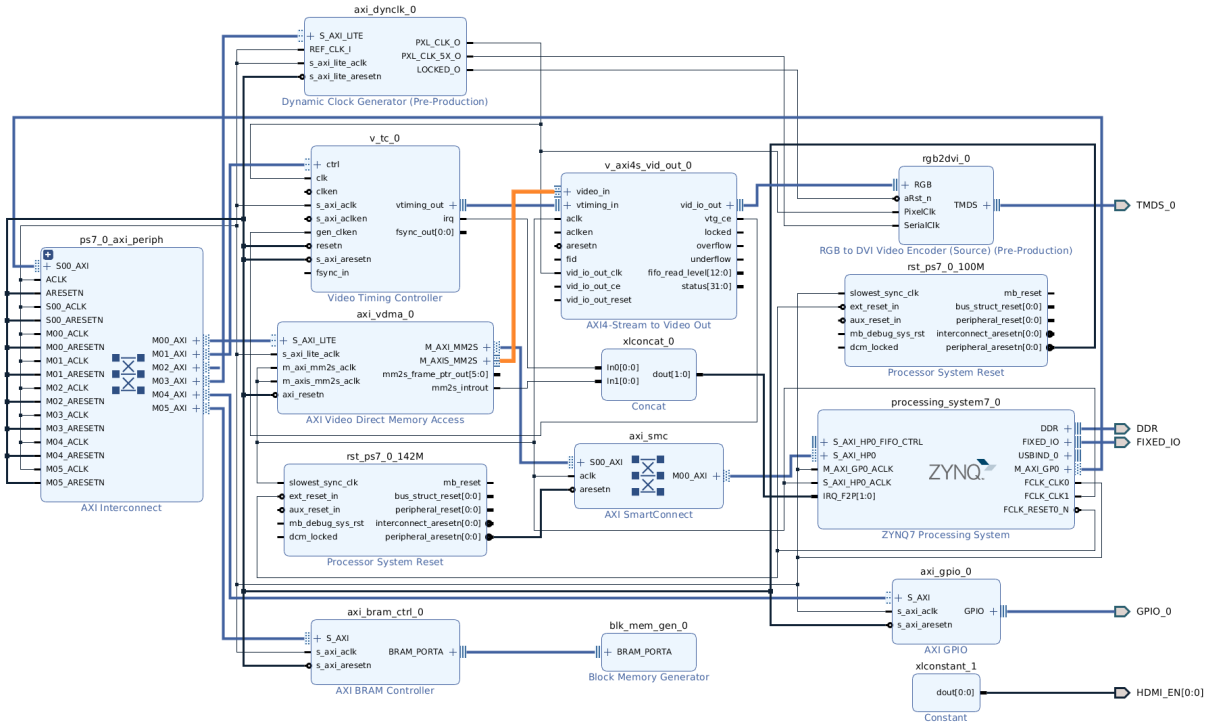
由于就是传统的内存访问,这里不展开说了,实际上,DMA在大规模数据传输的时候才是用途最广的,讲DMA之前先看一下框图.



看到PL外接口,包括AXI GP(PS 2CH M & PS 2CH S),AXO HP(PS 4CH S),AXI ACP(PS S),其中片内还有一个DMAC,由于属于ARM内核自身的,所以这里没写,它的理论速率是600MB/s,用户或ACP和HP接口的每一挑通道可以有1200MB/s,但是GP口依然是600MB/s,所以一般是GP口进行控制流,HP口作为数据流,其实DMA我们一直实验中都有用,就是VDMA,它其实是DMA的一个特殊形式,专门用于Video,如图高亮部分.



HP口通过AXI SmartConnect连接,主要是因为优化访问,凑齐了一并传输,也可以直连VDMA.它和Interconnect是兄弟,后者功能简单性能更低但是更节约资源.VDMA收到了数据,存好在DDR中,然后合适时候发送到AXI4-Stream to Video Out中,这里的数据流是原始的RGB数据,因此,可以在中间插入PL逻辑,自定义IP等等进行数据处理,比如二值化.



但是AXI4-Stream to Video Out出来的就是RGB时序了,当然它也需要输入一个RGB时钟,这个和DMA无关,另外我们还发现GP接口通过AXI Interconnect连接到Video Timing Controller,所以实际是通过GP接口配置了显示的时序的.在Linux上用of_dma_request_slave_channel获取病绑定DMA,然后使用dmaengine相关函数进行设置,这些对DMA的设置过程,实际上也是通过GP接口控制的.

另外由于挂在HP接口,实际上是由PL端主动发起访问至PS端,在实际应用中,用户通常在用户空间申请了一块内存,吭哧吭哧之后,希望交到PL处理,数据量少这没什么,数据量大的话,CPU那点复制效率是不堪重负的.如果把AXI DMA通过mmap方式在用户空间访问,虽然也能提高搬运效率,但是对于copy_to_user/copy_from_user还是没有半点用途,网上有人做了一个关于AXI DMA的零拷贝驱动,建议大家去了解一下,这里就不多说了.