Linux Uio Driver For AXI DMA

Liu,Yu-Tang

July 17, 2018

Outline

- Motivation and Preliminaries
- Problems
- Linux UIO For AXI DMA
- Analysis
- Conclusion

Motivation and Preliminaries

Motivation and Preliminaries

We want to develop software with our custom IP
with UIO driver in Linux on FPGA.

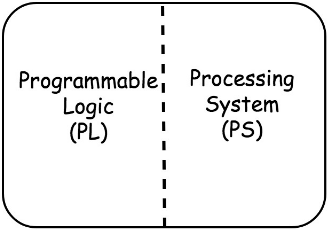
Motivation and Preliminaries

We want to develop software with our ***custom IP*** with ***UIO driver*** in ***Linux on FPGA***.

Motivation and Preliminaries

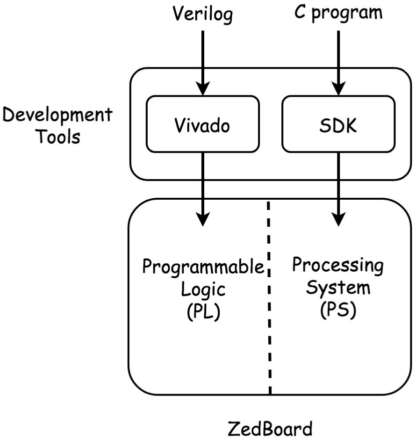
First of all, ***WHY?***

Motivation and Preliminaries



ZedBoard

Motivation and Preliminaries



Motivation and Preliminaries

There are some problems when we develop software in SDK.

- Some libraries are only in SDK.
- When burning program into FPGA, there may have some error, and is hard to debug.

Motivation and Preliminaries

If we can program in Linux on FPGA, the above problems can be solved.

- Some libraries are only in SDK.

Motivation and Preliminaries

If we can program in Linux on FPGA, the above problems can be solved.

- Some libraries are only in SDK.
  -> Standard libraries in Linux.

Motivation and Preliminaries

If we can program in Linux on FPGA, the above problems can be solved.

- Some libraries are only in SDK.
  -> Standard libraries in Linux.
- When burning program into FPGA, there may have some error, and is hard to debug.

Motivation and Preliminaries

If we can program in Linux on FPGA, the above problems can be solved.

- Some libraries are only in SDK.
  -> Standard libraries in Linux.
- When burning program into FPGA, there may have some error, and is hard to debug.
  ->Directly run the program in Linux.

Motivation

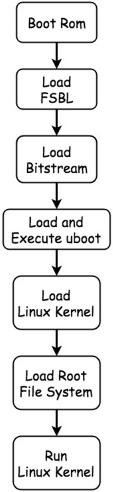
Motivation and Preliminaries

- Linux on FPGA
- Custom IP
- UIO Driver

Motivation and Preliminaries

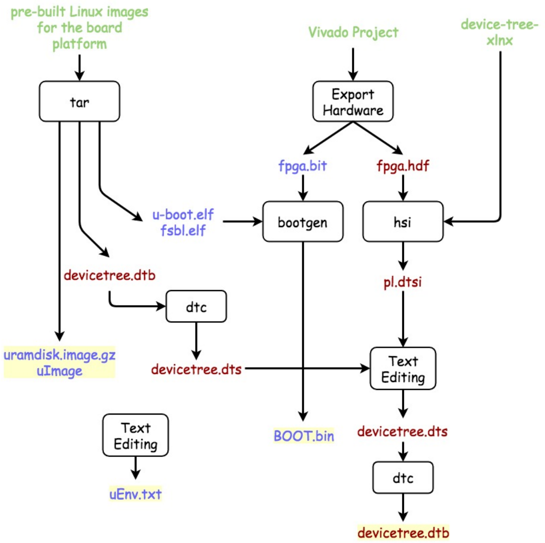
- Linux on FPGA
- Custom IP
- UIO Driver

Motivation and Preliminaries



We shows an example of booting Linux by SD card on FPGA.

Motivation and Preliminaries



Motivation and Preliminaries

So we need

- BOOT.bin
- devicetree.dtb
- uramdisk
- uImage

Motivation and Preliminaries

- BOOT.bin
  - fsbl.elf
  - u-boot.elf
  - fpga.bit

devicetree.dtb

- Device Tree is a mechanism to describe all hardware and devices of a system.

Motivation and Preliminaries

uramdisk.image.gz

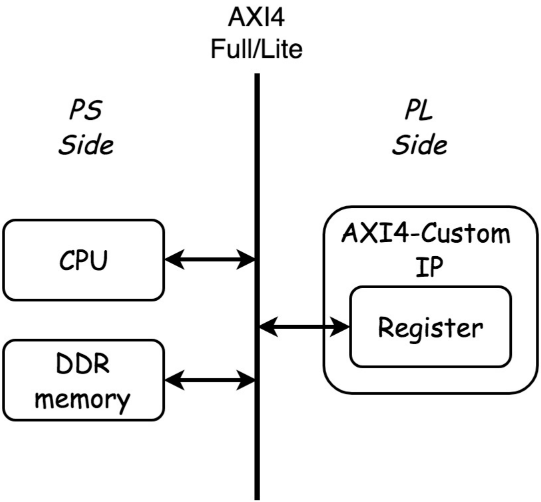
- Initialize RAM disk, not root file system uImage

- Linux Kernel with u-boot header.

Motivation and Preliminaries

- Linux on FPGA
- Custom IP
- UIO Driver

Motivation and Preliminaries



Motivation and Preliminaries

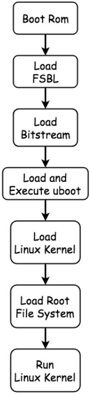
- Linux on FPGA
- Custom IP
- UIO Driver

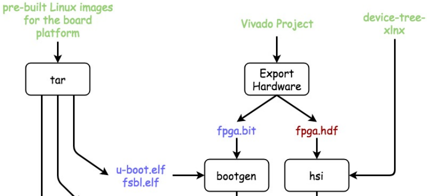
Motivation and Preliminaries

UIO Driver

- only one small kernel module to write and maintain.
- develop the main part of your driver in user space, with all the tools and libraries you're used to.
- bugs in your driver won't crash the kernel.
- updates of your driver can take place without recompiling the kernel.

Motivation and Preliminaries





Embedded Linux on zedboard

Embedded Linux on zedboard

- Phase 0
  - Load boot rom
  - Check JP7-JP11
- Phase 1
  - FSBL
- Phase 2
  - Software

Embedded Linux on zedboard

- So we need ....
  - Boot.bin
  - Uboot
  - devicetree
  - rootfs

Embedded Linux on zedboard

- Boot.bin
  - fsbl.elf
  - boot.elf
  - vivado-design.bit

Embedded Linux on zedboard

Uboot

- Das U-Boot (subtitled "the Universal Boot Loader" and often shortened to U-Boot) is an open source, primary boot loader used in embedded devices to package the instructions to boot the device's operating system kernel.

Embedded Linux on zedboard

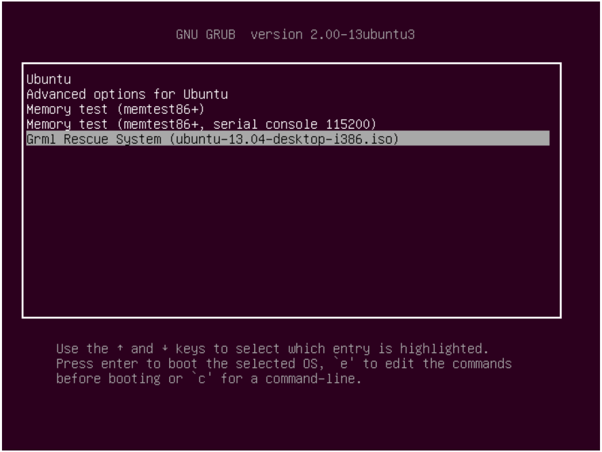


Figure: grub

Embedded Linux on zedboard

devicetree

- In computing, a device tree (also written devicetree) is a data structure describing the hardware components of a particular computer so that the operating system's kernel can use and manage those components, including the CPU or CPUs, the memory, the buses and the peripherals.

Embedded Linux on zedboard

```
        usb@e0003000 {
            compatible = "xlnx,zynq-usb-2.20a", "chipidea,usb2";
            status = "disabled";
            clocks = <0x1 0x1d>;
            interrupt-parent = <0x3>;
            interrupts = <0x0 0x2c 0x4>;
            reg = <0xe0003000 0x1000>;
            phy_type = "ulpi";
        };

        watchdog@f8005000 {
            clocks = <0x1 0x2d>;
            compatible = "cdns,wdt-r1p2";
            interrupt-parent = <0x3>;
            interrupts = <0x0 0x9 0x1>;
            reg = <0xf8005000 0x1000>;
            timeout-sec = <0xa>;
        };

        my_AES@43c00000 {
            compatible = "xlnx,my-AES-1.0";
            reg = <0x43c00000 0x10000>;
            interrupts = <0 29 1>;
            interrupt-parent = <0x3>;
            xlnx,s00-axi-addr-width = <0x6>;
            xlnx,s00-axi-data-width = <0x20>;
        };
    };

    phy0 {
        compatible = "ulpi-phy";
        #phy-cells = <0x0>;
        reg = <0xe0002000 0x1000>;
        view-port = <0x170>;
        drv-vbus;
        linux,phandle = <0x6>;
        phandle = <0x6>;
    };
};
```

Embedded Linux on zedboard

root file system

- The root filesystem is the filesystem that is contained on the same partition on which the root directory is located, and it is the filesystem on which all the other filesystems are mounted (i.e., logically attached to the system) as the system is booted up (i.e., started up).

Embedded Linux on zedboard

Now we can run linux on zedboard!

UIO driver

Uio Driver

About UIO

- only one small kernel module to write and maintain.
- develop the main part of your driver in user space, with all the tools and libraries you're used to.
- bugs in your driver won't crash the kernel.
- updates of your driver can take place without recompiling the kernel.

Uio Driver

How UIO works?

- Each UIO device is accessed through a device file and several sysfs attribute files. The device file will be called /dev/uio0 for the first device, and /dev/uio1, /dev/uio2 and so on for subsequent devices.
  /dev/uioX is used to access the address space of the card. Just use mmap() to access registers or RAM locations of your card.

Uio Driver

How to use UIO

```
int fd = open("/dev/uio0", O_RDWR);

void *ptr = mmap(0, 0x10000, PROT_READ|PROT_WRITE, MAP_SHAR

volatile uint32_t *ctrl = (uint32_t *)ptr;
volatile uint32_t *memA = (uint32_t *)ptr + 1;
```

Uio Driver

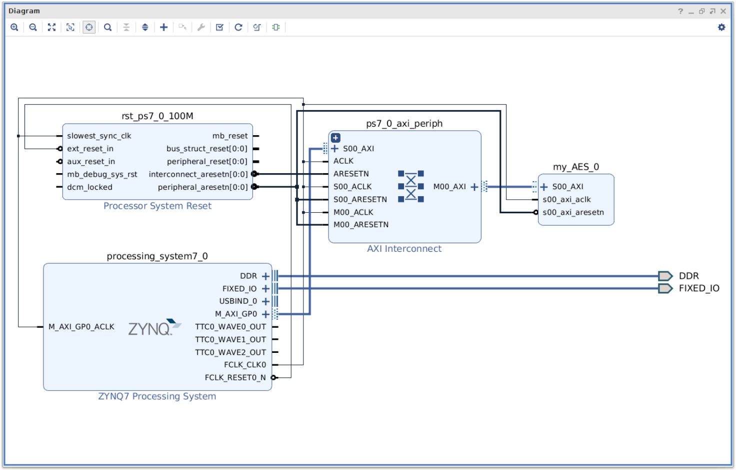


Diagram
rst_ps7_0_100M
slowest_sync_clk
ext_reset_in
aux_reset_in
mb_debug_sys_rst
dcm_locked
mb_reset
bus_struct_reset[0:0]
peripheral_reset[0:0]
interconnect_aresetn[0:0]
peripheral_aresetn[0:0]
Processor System Reset
ps7_0_axi_periph
S00_AXI
ACLK
ARESETN
S00_ACLK
S00_ARESETN
M00_ACLK
M00_ARESETN
M00_AXI
AXI Interconnect
my_AES_0
S00_AXI
s00_axi_aclk
s00_axi_aresetn
processing_system7_0
DDR
FIXED_IO
USBIND_0
M_AXI_GP0
M_AXI_GP0_ACLK
ZYNQ
TTC0_WAVE0_OUT
TTC0_WAVE1_OUT
TTC0_WAVE2_OUT
FCLK_CLK0
FCLK_RESET0_N
ZYNQ7 Processing System
DDR
FIXED_IO

Uio Driver

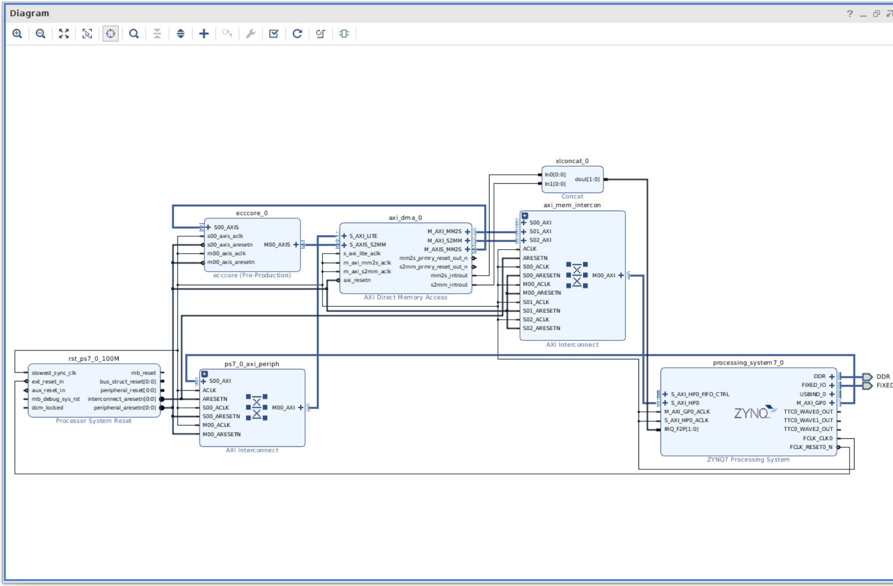


Diagram
xlconcat_0
In0[0:0]
In1[0:0]
dout[1:0]
Concat
ecccore_0
S00_AXIS
s00_axis_aclk
s00_axis_aresetn
m00_axis_aclk
m00_axis_aresetn
M00_AXIS
ecccore (Pre-Production)
axi_dma_0
S_AXI_LITE
S_AXIS_S2MM
s_axi_lite_aclk
m_axi_mm2s_aclk
m_axi_s2mm_aclk
axi_resetn
M_AXI_MM2S
M_AXI_S2MM
M_AXIS_MM2S
mm2s_prmry_reset_out_n
s2mm_prmry_reset_out_n
mm2s_introut
s2mm_introut
AXI Direct Memory Access
axi_mem_intercon
S00_AXI
S01_AXI
S02_AXI
ACLK
ARESETN
S00_ACLK
S00_ARESETN
M00_ACLK
M00_ARESETN
S01_ACLK
S01_ARESETN
S02_ACLK
S02_ARESETN
M00_AXI
AXI Interconnect
rst_ps7_0_100M
slowest_sync_clk
ext_reset_in
aux_reset_in
mb_debug_sys_rst
dcm_locked
mb_reset
bus_struct_reset[0:0]
peripheral_reset[0:0]
interconnect_aresetn[0:0]
peripheral_aresetn[0:0]
Processor System Reset
ps7_0_axi_periph
S00_AXI
ACLK
ARESETN
S00_ACLK
S00_ARESETN
M00_ACLK
M00_ARESETN
M00_AXI
AXI Interconnect
processing_system7_0
S_AXI_HP0_FIFO_CTRL
S_AXI_HP0
M_AXI_GP0_ACLK
S_AXI_HP0_ACLK
IRQ_F2P[1:0]
DDR
FIXED_IO
USBIND_0
M_AXI_GP0
TTC0_WAVE0_OUT
TTC0_WAVE1_OUT
TTC0_WAVE2_OUT
FCLK_CLK0
FCLK_RESET0_N
ZYNQ
ZYNQ7 Processing System
DDR
FIXED

DMA and AXI4

DMA and AXI4

DMA

- Direct memory access (DMA) is a feature of computer systems that allows certain hardware subsystems to access main system memory (Random-access memory), independent of the central processing unit (CPU).

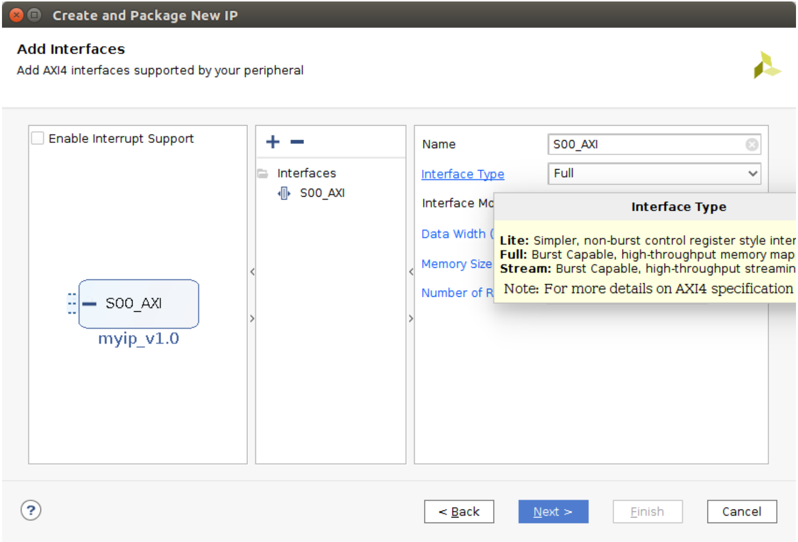
DMA and AXI4

AXI4

- AMBA® AXI4 (Advanced eXtensible Interface 4) is the fourth generation of the AMBA interface specification from ARM®. Xilinx Vivado Design Suite 2014 and ISE Design Suite 14 extends the Xilinx platform design methodology with the semiconductor industry's first AXI4 Compliant Plug-and-Play IP.

DMA and AXI4

register type



DMA and AXI4

- AXI4
  - Traditional memory mapped address/data interface.
  - Data burst support.
- AXI4-Lite
  - Traditional memory mapped address/data interface.
  - Single data cycle only.
- AXI4-stream
  - Data-only burst.

DMA and AXI4

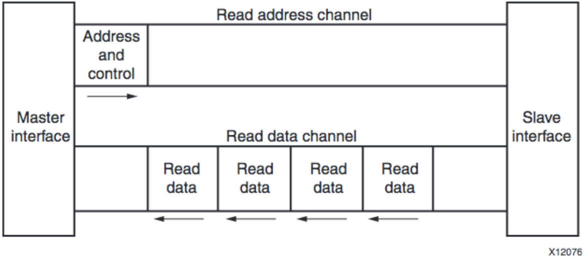


Figure 1-1: **Channel Architecture of Reads**

DMA and AXI4

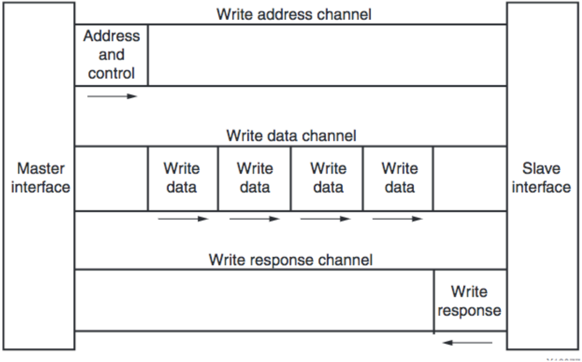
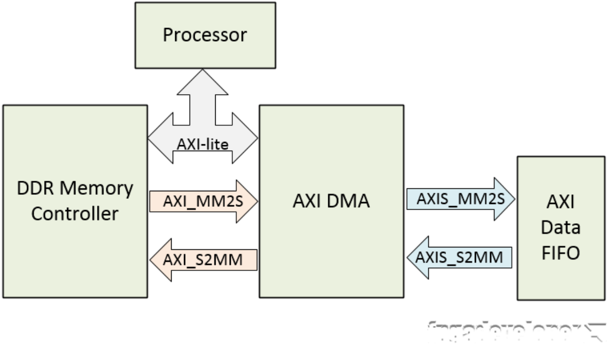


Figure 1-2: **Channel Architecture of Writes**

DMA and AXI4


Processor
AXI-lite
DDR Memory Controller
AXI_MM2S
AXI_S2MM
AXI DMA
AXIS_MM2S
AXIS_S2MM
AXI Data FIFO

DMA and AXI4

- If our custom IP uses AXI-Stream registers, we can't use UIO driver to mount our IP as a device.

DMA and AXI4

But we still want to use our IP in userspace!

DMA and AXI4

But we still want to use our IP in userspace!

->UIO driver modification

UIO driver modification

UIO driver modification

In fact, DMA component is driven by the AXI-DMA module provided by xilinx linux kernel.
So we can use Linux Kernel DMA API in our UIO driver.

UIO driver modification

- DMA API
  - dma_request_slave_channel
  - dmaengine_prep_slave_sg
  - dmaengine_submit

UIO driver modification

We can create a virtual device node to apply UIO driver.

UIO driver modification

devicetree will be like:

```
        xlnx,include-sg;
        loopback_dma_mm2s_chan: dma-channel@40410000 {
            compatible = "xlnx,axi-dma-mm2s-channel";
            interrupt-parent = <&gic>;
            interrupts = <0 31 4>;  // concat port 2
                                    // IRQ_F2P[15:0] == [91:84],[68:61]
                                    // 2 -> 63 -> 63 - 32 = 31

            xlnx,datawidth = <0x20>;        // 32-bit output
            xlnx,sg-length-width = <14>;    // Width of Buffer Length Register (configured for 20 bits)

            xlnx,device-id = <0x1>;     // what's this for?
        };

        loopback_dma_s2mm_chan: dma-channel@40410030 {
            compatible = "xlnx,axi-dma-s2mm-channel";
            interrupt-parent = <&gic>;
            interrupts = <0 32 4>;  // concat port 3
                                    // IRQ_F2P[15:0] == [91:84],[68:61]
                                    // 3 -> 64 -> 64 - 32 = 32

            xlnx,datawidth = <0x20>;        // 32-bit output
            xlnx,sg-length-width = <14>;    // Width of Buffer Length Register (configured for 20 bits)

            xlnx,device-id = <0x1>;     // what's this for?
        };
    };

    udma0 {
        compatible = "generic-uio";

        dmas = <&loopback_dma 0 &loopback_dma 1>;
        dma-names = "loop_tx", "loop_rx";   // used when obtaining reference to above DMA core using dma_request_slave_channel()
        ezdma,dirs = <2 1>;                 // direction of DMA channel: 1 = RX (dev->cpu), 2 = TX (cpu->dev)
    };
```

UIO driver modification

recall

```
int fd = open("/dev/uio0", O_RDWR);

void *ptr = mmap(0, 0x10000, PROT_READ|PROT_WRITE, MAP_SHAR
```

UIO driver modification

read()

- Interrupts are handled by reading from /dev/uioX. A blocking read() from /dev/uioX will return as soon as an interrupt occurs. You can also use select() on /dev/uioX to wait for an interrupt. The integer value read from /dev/uioX represents the total interrupt count. You can use this number to figure out if you missed some interrupts.

UIO driver modification

write()

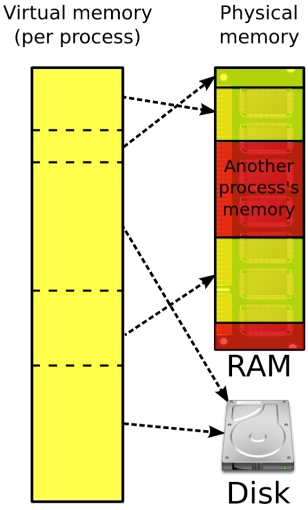
- UIO also implements a write() function,a write() to /dev/uioX will call the irqcontrol() function implemented by the driver.You have to write a 32-bit value that is usually either 0 or 1 to disable or enable interrupts. If a driver does not implement irqcontrol(), write() will return with -ENOSYS.

UIO driver modification

Virtual memory

- The addresses a program may use to reference memory are distinguished from the addresses the memory system uses to identify physical storage sites, and program generated addresses are translated automatically to the corresponding machine addresses.

UIO driver modification

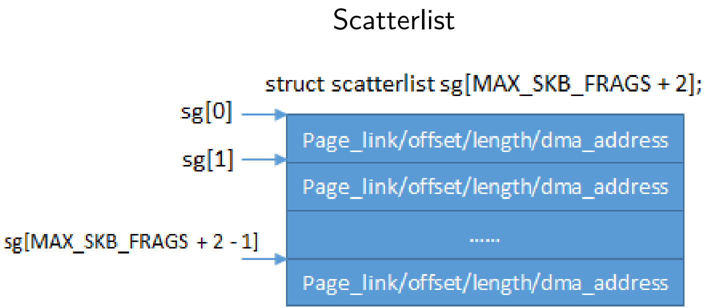


UIO driver modification

Scatter-Gather

Scatter-Gather DMA augments this technique by providing data transfers from one non-contiguous block of memory to another by means of a series of smaller contiguous-block transfers. The Lattice Scatter-Gather DMA Controller core implements a configurable, multi-channel, WISHBONE-compliant DMA controller with scatter-gather capability.

UIO driver modification



Thanks!