Vaasan yliopisto
UNIVERSITY OF VAASA

ICAT3170, SOC-FPGA

How to use DMA from Linux

Updated version 24.3.2020

Why Linux?

Vaasan yliopisto
UNIVERSITY OF VAASA

- It has millions of apps (= SW)
- It has POSIX programming interface
- It provides operating systems services: Dynamic memory allocation, processes, priorities, file system, TCP/IP, web server, SSH, FTP, …
- Application development is fast
- It can be self hosted: vim + gcc + make
- It can be tailored to be very light and efficient (32 M memory footprint, 20 MB file system)

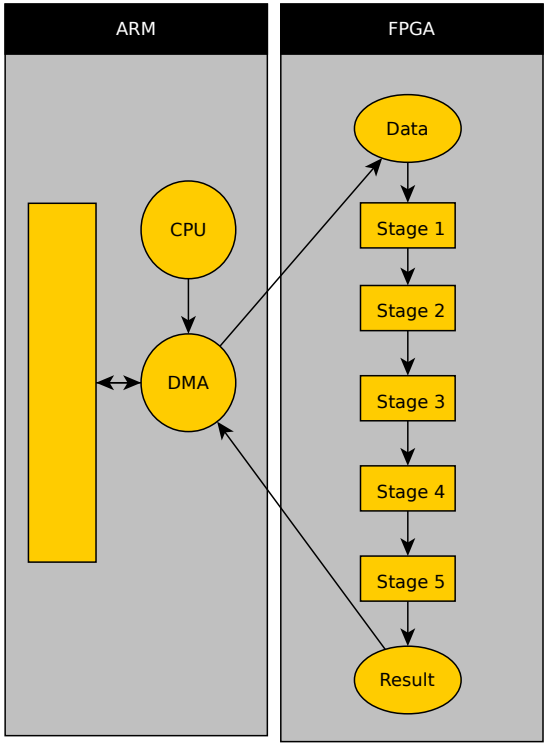
```
root@dmakoe-nfm:~# free
             total       used       free     shared    buffers     cached
Mem:       1030364      32720     997644      15104        164      15104
-/+ buffers/cache:      17452    1012912
Swap:            0          0          0
root@dmakoe-nfm:~#
```

- Why not?
  - It is not exactly real time
  - Virtual memory can make low level things complicated
  - The development system is more complicated
  - Linux knowledge is needed

Pipelined HW acceleration.

Vaasan yliopisto
UNIVERSITY OF VAASA

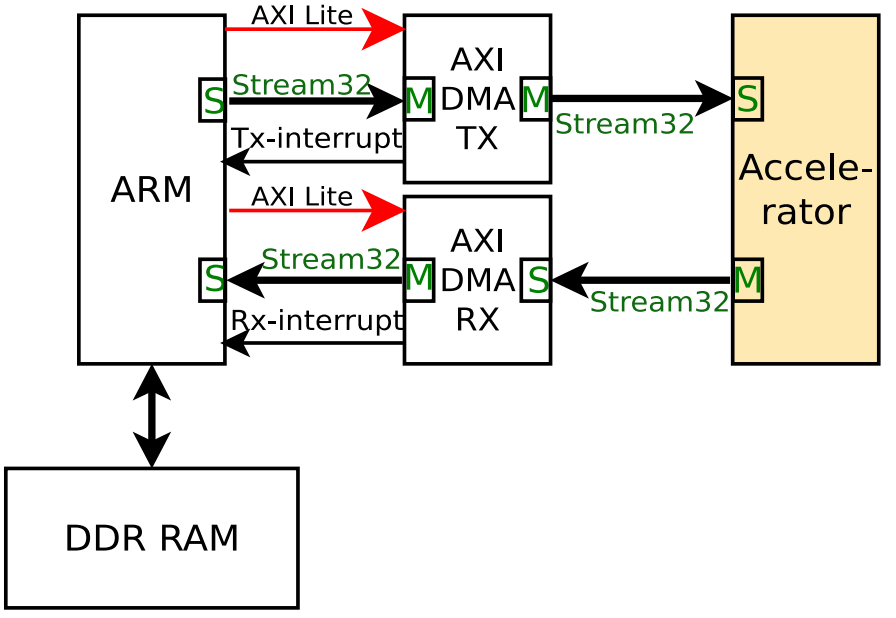
- Efficient image processing accelerator can be implemented as a pipeline
- The DMA feeds data in the first end of the pipeline
- Another DMA channels takes the data back to the CPU after processing
- The pipeline does not fetch the data from RAM or write it it there
- It only processes the data
- Pipelines can be pipelined.



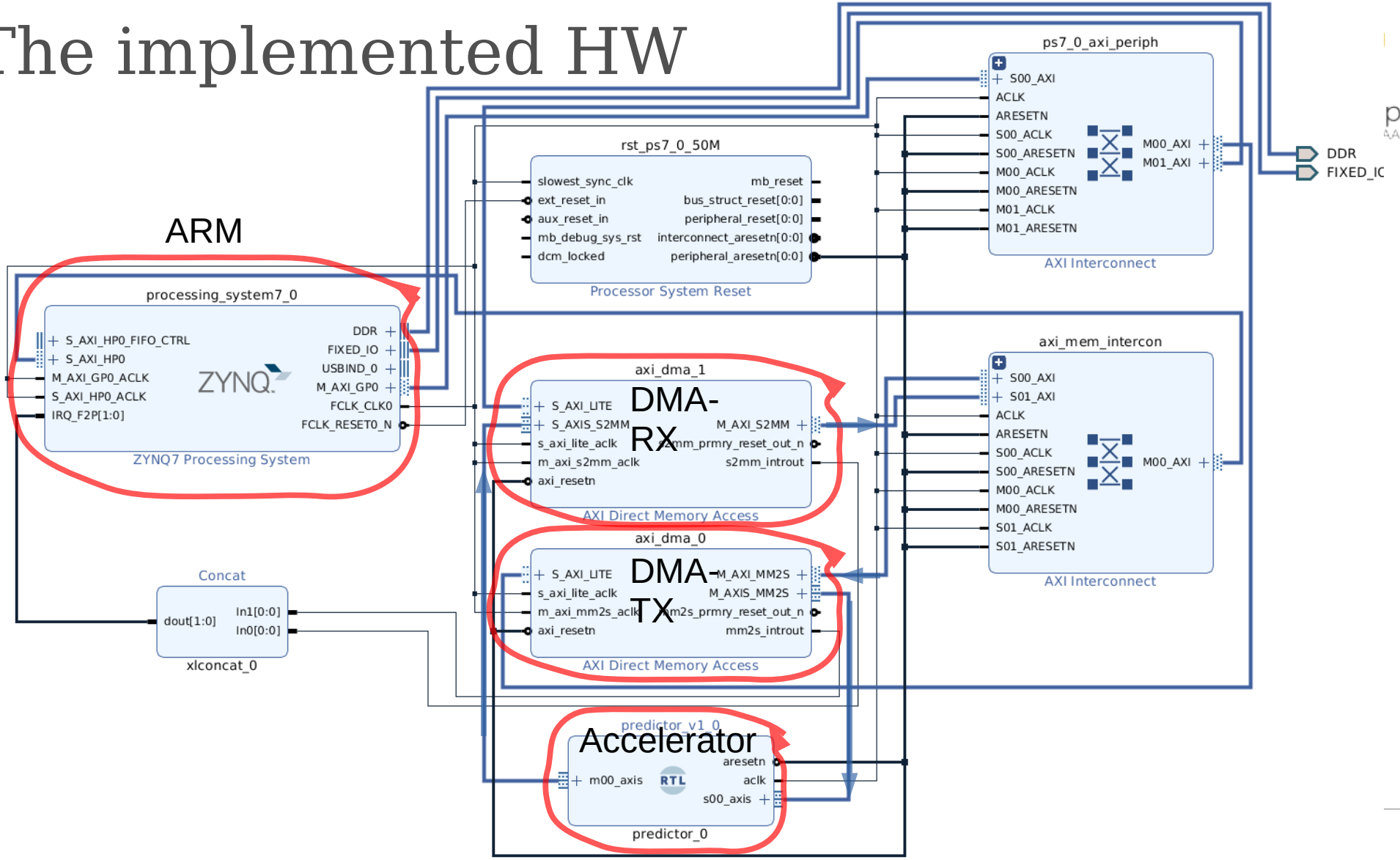
Communication through a DMA

Vaasan yliopisto
UNIVERSITY OF VAASA

- DMA is configured from the ARM processing system by control registers through the AXI Lite interface
- DMA reads and writes data to the DDR RAM using high performance (200 MB/s) AXI stream buses without bothering the processing system
- The DMA also reads and writes data to the accelerator component using similar AXI stream buses
- When tx and rx transmits are ready, the DMA sends an interrupt to the processing unit to inform that the task is finished. Therefore the ARM does not need to poll to check when the transfer is finished.



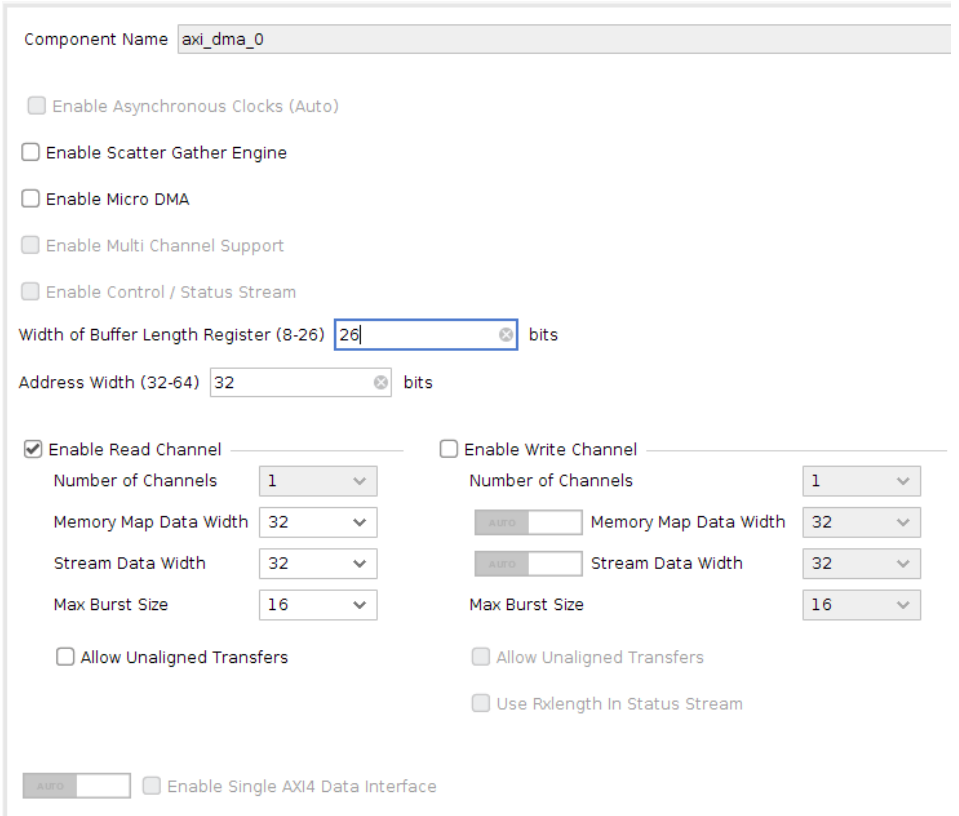
The implemented HW



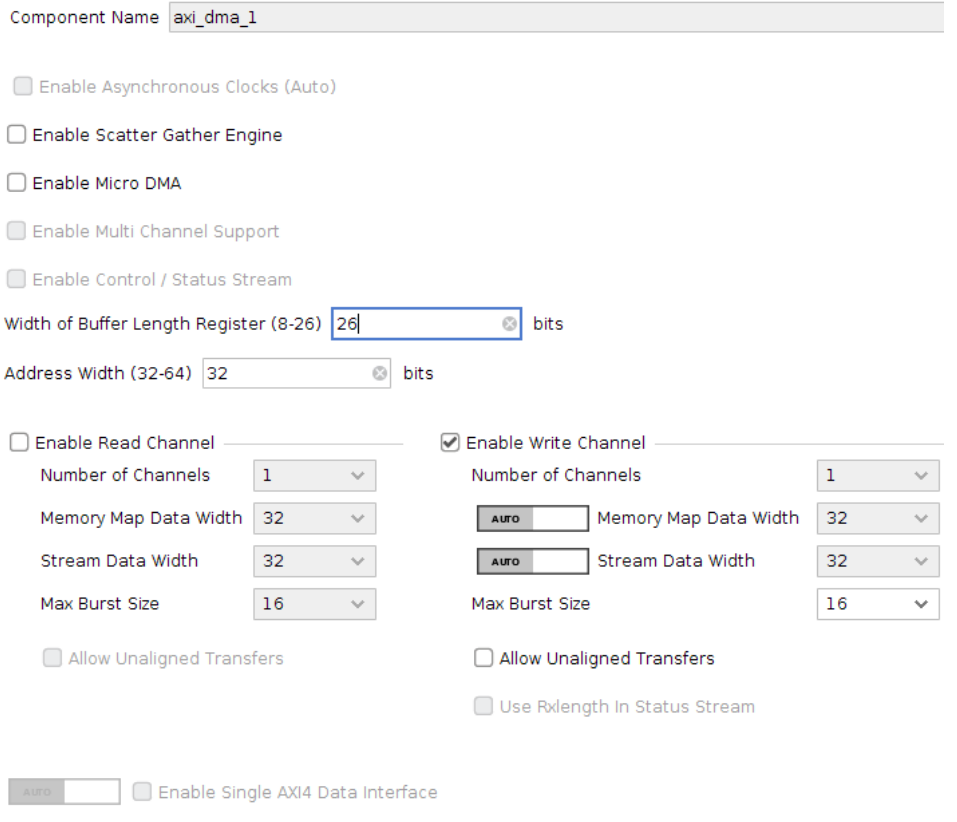
AXI-DMA configurations

Vaasan yliopisto
UNIVERSITY OF VAASA

TX AXI DMA



RX AXI DMA



Memory map

Vaasan yliopisto
UNIVERSITY OF VAASA

- The base address of the DMA control register is 0x4040_0000
- AXI-DMAs have their own memory spaces in their buses, which can be either 64-bit or 32-bits, here they are 32 bits wide.
-

Diagram | Address Editor

Cell	Slave Interface	Slave Segment	Offset Address	Range	High Address
processing_system7_0					
Data (32 address bits : 0x40000000 [1G])					
axi_dma_0	S_AXI_LITE	Reg	0x4040_0000	64K	0x4040_FFFF
axi_dma_1	S_AXI_LITE	Reg	0x4041_0000	64K	0x4041_FFFF
axi_dma_0					
Data_MM2S (32 address bits : 4G)					
processing_system7_0	S_AXI_HP0	HP0_DDR_LOWOCM	0x0000_0000	1G	0x3FFF_FFFF
axi_dma_1					
Data_S2MM (32 address bits : 4G)					
processing_system7_0	S_AXI_HP0	HP0_DDR_LOWOCM	0x0000_0000	1G	0x3FFF_FFFF

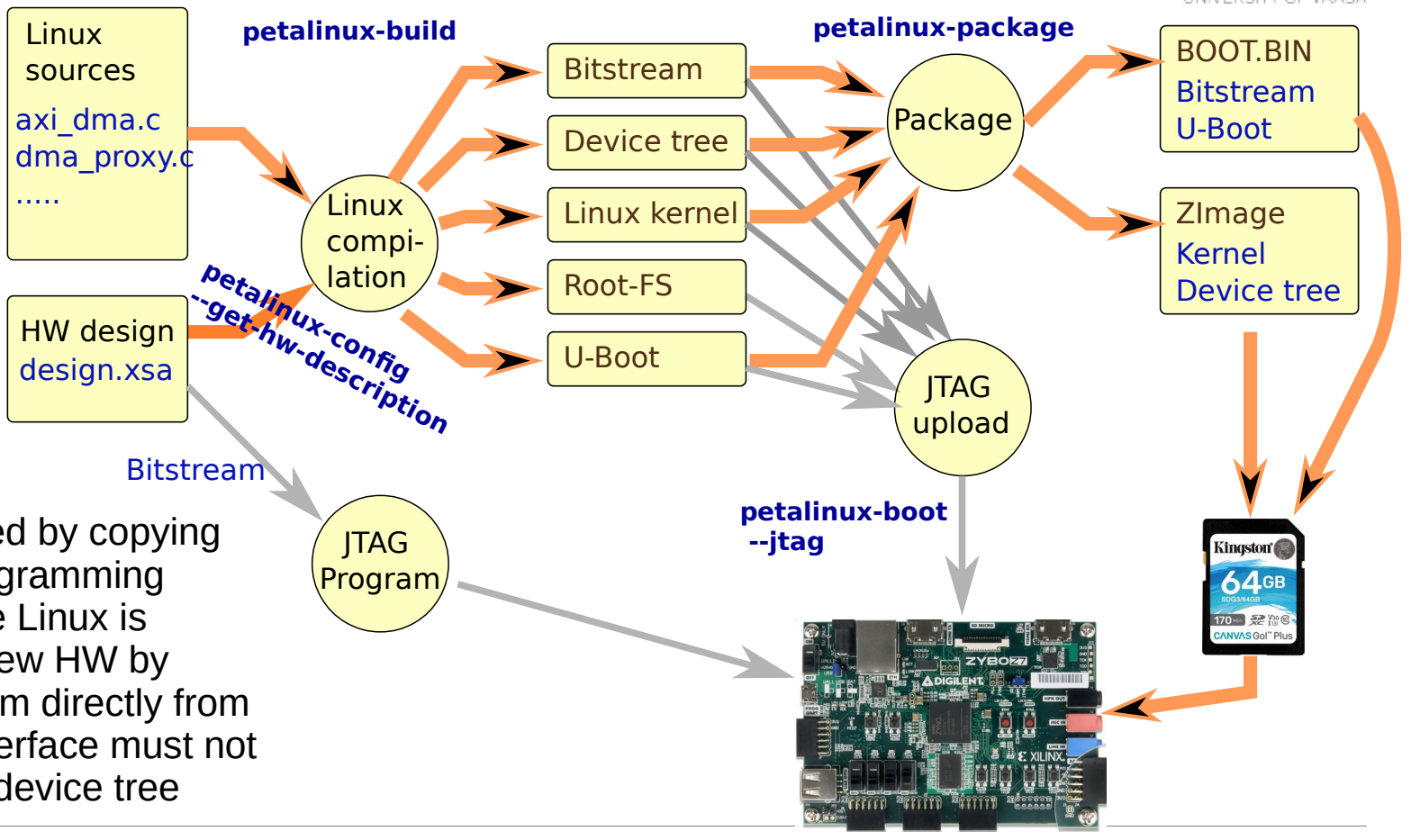
Registers

- The DMA registers from AXI_DMA component documentation
- The addresses are relative to the base address, eg 0x4040_0000
- The configuration of these register are handled by the board support package or by Linux

Table 2-5: **Scatter / Gather Mode Register Address Map**

Address Space Offset[1]	Name	Description
00h	MM2S_DMACR	MM2S DMA Control register
04h	MM2S_DMASR	MM2S DMA Status register
08h	MM2S_CURDESC	MM2S Current Descriptor Pointer. Lower 32 bits of the address.
0Ch	MM2S_CURDESC_MSB	MM2S Current Descriptor Pointer. Upper 32 bits of address.
10h	MM2S_TAILDESC	MM2S Tail Descriptor Pointer. Lower 32 bits.
14h	MM2S_TAILDESC_MSB	MM2S Tail Descriptor Pointer. Upper 32 bits of address.
2Ch [2]	SG_CTL	Scatter/Gather User and Cache
30h	S2MM_DMACR	S2MM DMA Control register
34h	S2MM_DMASR	S2MM DMA Status register
38h	S2MM_CURDESC	S2MM Current Descriptor Pointer. Lower 32 address bits
3Ch	S2MM_CURDESC_MSB	S2MM Current Descriptor Pointer. Upper 32 address bits.
40h	S2MM_TAILDESC	S2MM Tail Descriptor Pointer. Lower 32 address bits.
44h	S2MM_TAILDESC_MSB	S2MM Tail Descriptor Pointer. Upper 32 address bits.

Compilation process

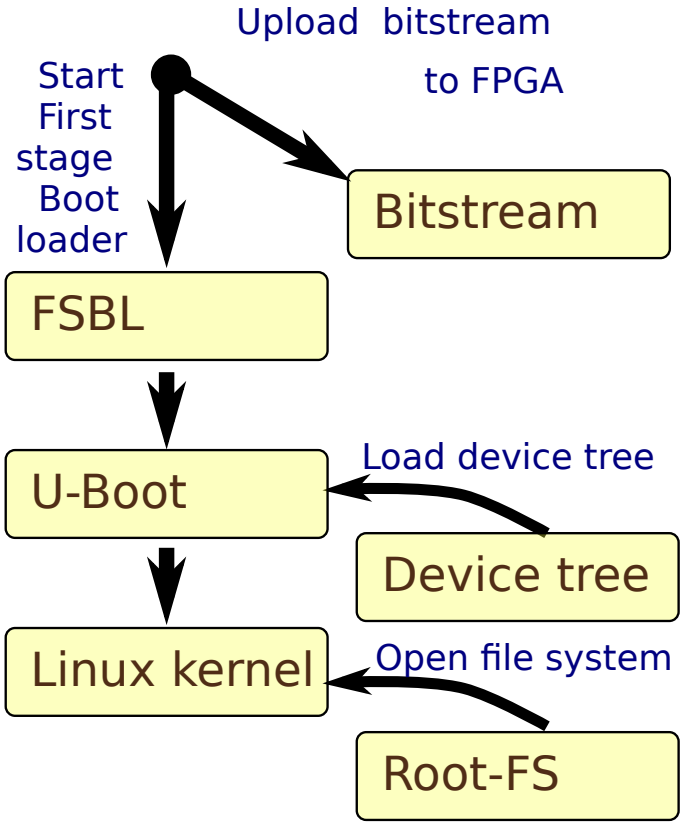


The system can be installed by copying the files in SD card, or programming them with JTAG. When the Linux is running, you can update new HW by programming new Bitstream directly from vivado. In this case the interface must not change, so that the same device tree works.

Linux booting process in FPGA

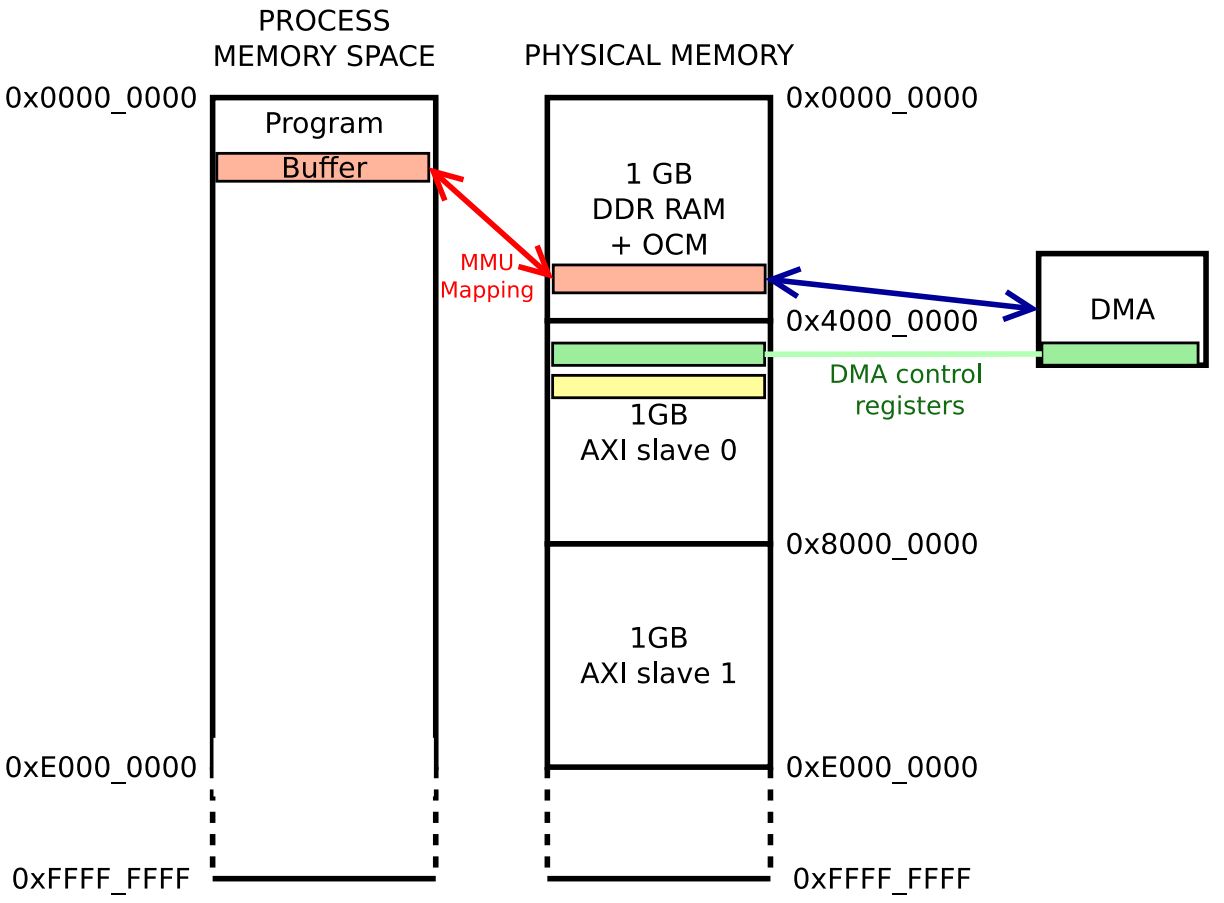
Vaasan yliopisto
UNIVERSITY OF VAASA

- Before booting, all required components needs to be loaded into the FPGA board
- Especially there needs to be a bitstream before starting the bootloader
- The first stage bootloader starts embedded linux bootloader: U-Boot
- U-Boot loads the device tree and starts the linux kernel
- The kernel opens the compressed root file system image and starts the system by executing the init scripts from the root file system
- Finally it configures the network and starts and SSH-server, to allow login over TCP/IP
- Bitstream can be replaced after linux is booted, provided that it does not require changes in the device tree
- New software can be cross-compiled in the host computer and uploaded into the running system over SSH



Virtual memory

- Each process have their private memory space in virtual memory systems
- DMA transfer needs buffers, whose location in the physical memory is known and locked
- The buffers needs to be cache coherent
- Therefore, DMA transfer needs preparations in kernel space: a kernel driver is needed



Xilinx AXI DMA driver

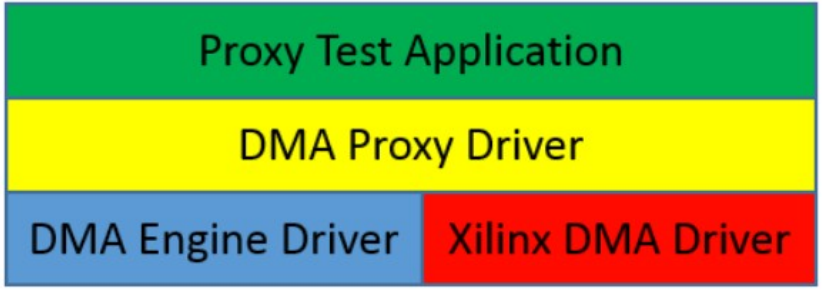
Vaasan yliopisto
UNIVERSITY OF VAASA

- Zero-copy transmit (processor to FPGA), receive (FPGA to processor), and two-way combined DMA transfers. (But there seems to be some issues with two way communication, it is therefore better to use two separate AXI DMA components)
- Support for transfers with Xilinx's AXI DMA and AXI VDMA (video DMA) IP blocks.
- Allocation of DMA buffers that are contiguous in physical memory, allowing for high-bandwidth DMA transfers, through the kernel's contiguous memory allocator (CMA).
- Allocation of memory that is coherent between the FPGA and processor, by disabling caching for those pages in the DMA buffer.
- Synchronous and asynchronous modes for transfers.
- Registration of callback functions that are called when an asynchronous transfer completes.
- Delivery of a POSIX real-time signal upon completion of an asynchronous transfer.
- Support for DMA buffer sharing, or external DMA buffers. Currently the driver can only import a DMA buffer from another driver. This is useful, for example, when transfers need to be done with a frame buffer allocated by a DRM driver.

DMA proxy driver

Vaasan yliopisto
UNIVERSITY OF VAASA

- The Xilinx axi_dma driver is already included in the standard linux kernel
- The linux DMA Engine provides convenient abstractions for using the DMA driver, but it is still rather low level
- Xilinx DMA-Proxy driver provides simple interface to user space applications, but it needs to be tailored for each purpose
- This is the way we use it here



Device tree entry for axi dma and proxy

Vaasan yliopisto
UNIVERSITY OF VAASA

```
dma@40400000 {
        #dma-cells = < 0x01 >;
        clock-names = "s_axi_lite_aclk\0m_axi_mm2s_aclk";
        clocks = < 0x01 0x0f 0x01 0x0f >;
        compatible = "xlnx,axi-dma-7.1\0xlnx,axi-dma-1.00.a";
        interrupt-names = "mm2s_introut";
        interrupt-parent = < 0x04 >;
        interrupts = < 0x00 0x1e 0x04 >;
        reg = < 0x40400000 0x10000 >;
        xlnx,addrwidth = < 0x20 >;
        xlnx,sg-length-width = < 0x1a >;
        phandle = < 0x07 >;

        dma-channel@40400000 {
                compatible = "xlnx,axi-dma-mm2s-channel";
                dma-channels = < 0x01 >;
                interrupts = < 0x00 0x1e 0x04 >;
                xlnx,datawidth = < 0x20 >;
                xlnx,device-id = < 0x00 >;
        };
};
```

There is similar entry for RX DMA in the device tree, which phandle is 0x08

Write

```
dma_proxy {
        compatible = "xlnx,dma_proxy";
        dmas = < 0x07 0x00 0x08 0x00 >;
        dma-names = "dma_proxy_tx\0dma_proxy_rx";
};
```

/dev/dma_proxy_tx
/dev/dma_proxy_rx

DMA-Proxy driver

Vaasan yliopisto
UNIVERSITY OF VAASA

- axi_dma driver implements most low level parts of the DMA communication
- The proxy driver takes care of tasks which still needs to be implemented in the kernel context (not possible in user space)
- These are the the reservation of coherent memory (not using cache memories) from the physical memory space

```
static int mmap(struct file *file_p, struct vm_area_struct *vma)
{
  struct dma_proxy_channel *pchannel_p = (struct dma_proxy_channel *)file_p->private_data;

  return dma_mmap_coherent(pchannel_p->dma_device_p, vma,
              pchannel_p->interface_p, pchannel_p->interface_phys_addr,
              vma->vm_end - vma->vm_start);
}
```

User space, some relevant lines

Vaasan yliopisto
UNIVERSITY OF VAASA

- User space program can access the DMA by opening the device files for RX and TX channels and mapping them in the memory
- The transfer is started by simple IOCTL command
- The buffer locations and sizes are set by the header file share with the driver and the user space program

```
tx_proxy_fd = open("/dev/dma_proxy_tx", O_RDWR);
rx_proxy_fd = open("/dev/dma_proxy_rx", O_RDWR);

tx_proxy_interface_p = (struct dma_proxy_channel_interface *)
  mmap(NULL, sizeof(struct dma_proxy_channel_interface),
  PROT_READ | PROT_WRITE, MAP_SHARED, tx_proxy_fd, 0);

rx_proxy_interface_p = (struct dma_proxy_channel_interface *)
      mmap(NULL, sizeof(struct dma_proxy_channel_interface),
      PROT_READ | PROT_WRITE, MAP_SHARED, rx_proxy_fd, 0);


ioctl(rx_proxy_fd, 0, &dummy);
ioctl(tx_proxy_fd, 0, &dummy);
```

- The DMA device access details are provided by the device tree

Create Petalinux project

Vaasan yliopisto
UNIVERSITY OF VAASA

The following directory structure is assumed:

../predictor/predictor-hw (here is Vivado project)
../predictor/predictor-sw (here we make Petalinux project)
../predictor/sources (some provided source files)

If you have different directory structure,change the red parts

1) Create an empty project (from directory **../predictor**)
   petalinux-create --type project --template zynq --name predictor-sw

2) Get current HW description from Vivado project (.xsa) into the Petalinux project. Repeat this step when you update the HW
   cd predictor-sw
   petalinux-config –get-hw-description=../predictor-hw

Add DMA module and test application

Vaasan yliopisto
UNIVERSITY OF VAASA

- We can already compile the project that was made, but we want to add a dma_proxy module in it to make the use of DMA easier. Let's add also a dma_proxy_test program to test the DMA

1) First create and enable dma_proxy kernel module
   petalinux-create -t modules -n dma-proxy --enable
2) Then create and enable the test program
   petalinux-create -t apps -n dma-proxy-test --enable

- Copy the example source files in place of automatically created files:
  cp ../sources/dma-proxy/* project-spec/meta-user/recipes-modules/dma-proxy/files/

  cp ../sources/dma-proxy-test/* project-spec/meta-user/recipes-apps/dma-proxy-test/files/

Add DMA module and test application (p2)

Vaasan yliopisto
UNIVERSITY OF VAASA

- Also modify the following BitBake files (.bb)
  project-spec/meta-user/recipes-modules/dma-proxy/dma-proxy.bb
  project-spec/meta-user/recipes-apps/dma-proxy-test/dma-proxy.bb
- Otherwise, the header file is not found during compilation

```
SUMMARY = "Recipe for  build an external dma-proxy Linux kernel module"
SECTION = "PETALINUX/modules"
LICENSE = "GPLv2"
LIC_FILES_CHKSUM = "file://COPYING;md5=12f884d2ae1ff87c09e5b7ccc2c4ca7e"

inherit module

SRC_URI = "file://Makefile \
           file://dma-proxy.c \
           file://dma-proxy.h \
           file://COPYING \
          "

S = "${WORKDIR}"
```

Add entry in the device tree source

Vaasan yliopisto
UNIVERSITY OF VAASA

- The process automatically inserts all needed device tree entries for the HW in the Vivado project
- But not for the dma-proxy driver, which was added directly in the Petalinux project
- If there is no device three entry, the kernel driver does nothing.
- Therefore, modify the user device tree source file:
  project-spec/meta-user/recipes-bsp/device-tree/files/system-user.dtsi

```
/include/ "system-conf.dtsi"
/ {
        dma_proxy {
                        compatible ="xlnx,dma_proxy";
                        dmas = <&axi_dma_0 0
                                &axi_dma_1 0>;
                        dma-names = "dma_proxy_tx", "dma_proxy_rx";
                };
};
```

Booting with JTAG

Vaasan yliopisto
UNIVERSITY OF VAASA

- In development phase, it is quite convenient to upload the new design in the board using JTAG
- The command shown below, uploads the bitstream, first stage boot loader, U-Boot, device tree, and the linux kernel into the system and boots it
- You can specify another bitstream, device tree or kernel file if you want
- Check details with **petalinux-boot --help**

```
petri@kryyni:predictor-sw$ petalinux-boot --jtag --kernel --fpga

INFO: Configuring the FPGA...
INFO: Downloading bitstream: /lacie/xilinx/linux/predictor/predictor-sw/images/linux/system.bit to the target.
INFO: Downloading ELF file: /lacie/xilinx/linux/predictor/predictor-sw/images/linux/zynq_fsbl.elf to the target.
INFO: Downloading ELF file: /lacie/xilinx/linux/predictor/predictor-sw/images/linux/u-boot.elf to the target.
INFO: Loading image: /lacie/xilinx/linux/predictor/predictor-sw/images/linux/system.dtb at 0x08008000
INFO: Loading image: /lacie/xilinx/linux/predictor/predictor-sw/images/linux/zImage at 0x00008000

INFO: SOC Silicon version is 3.1.
```

Links

Vaasan yliopisto
UNIVERSITY OF VAASA

- Xilinx Wiki
- Xilinx Linux drivers
- The Device tree
- Device tree generator for Vivado
- Xilinx AXI DMA v7.1 IP documentation

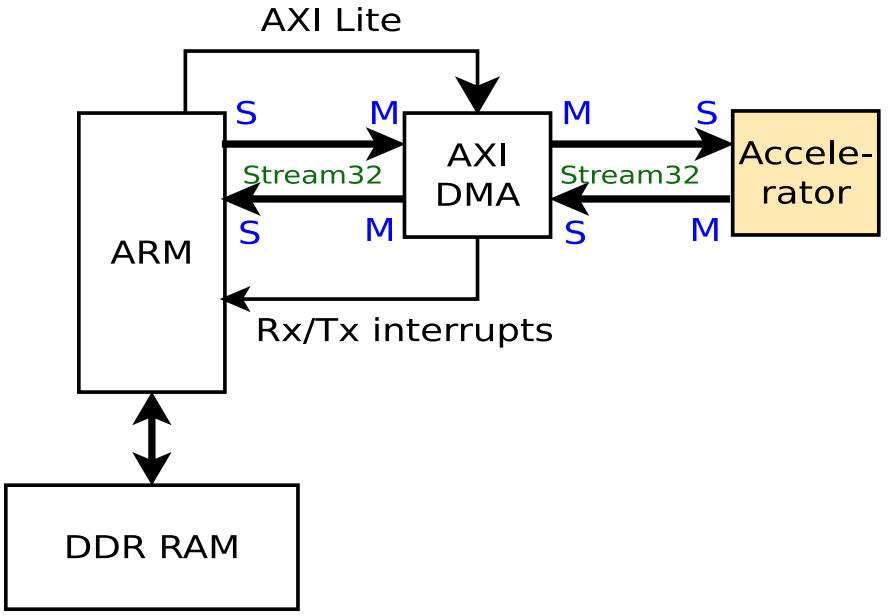
Vaasan yliopisto
UNIVERSITY OF VAASA

Old Stuff
This may work for some purposes, but
Probably not with Linux axi_dma driver!

Communication through a DMA

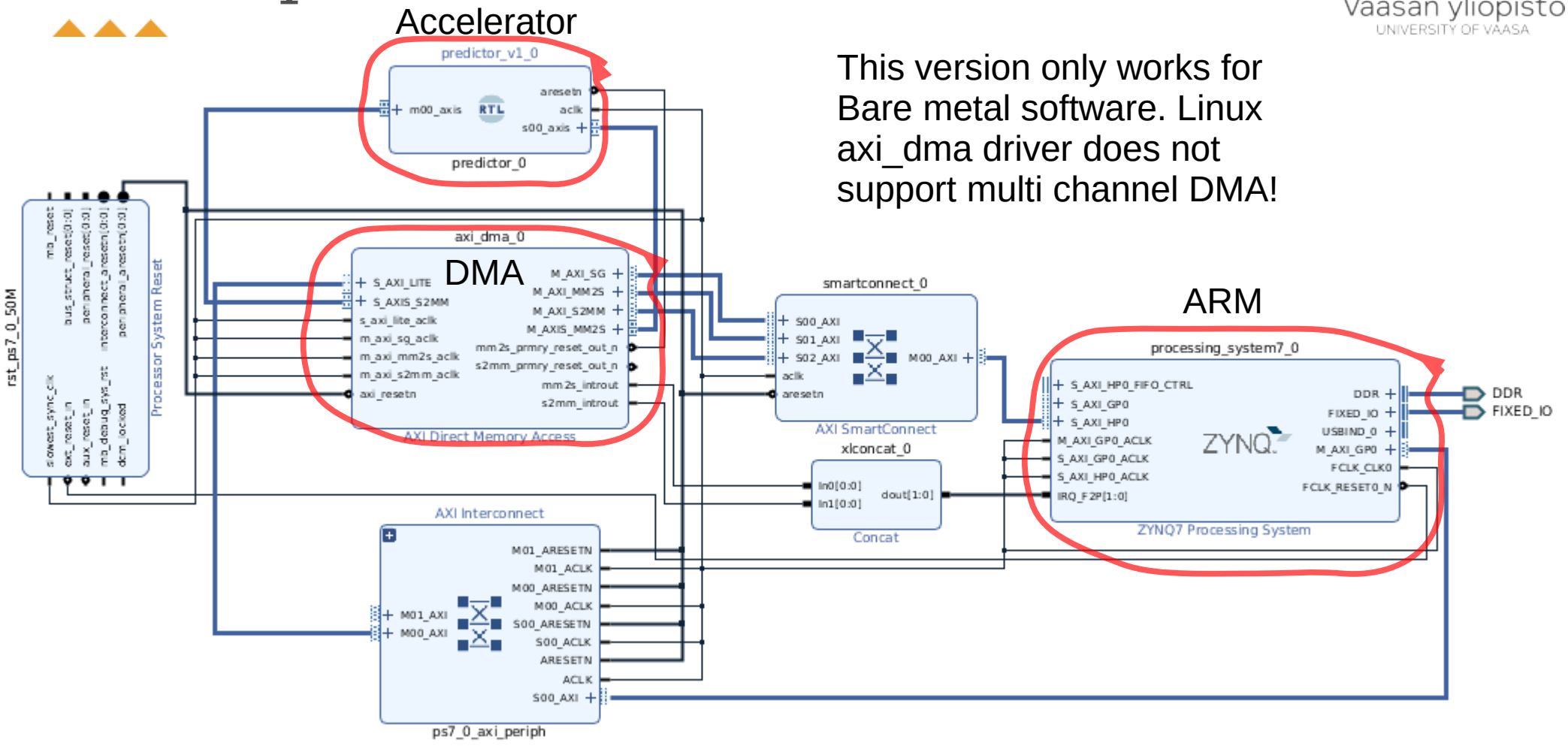
Vaasan yliopisto
UNIVERSITY OF VAASA

- DMA is configured from the ARM processing system by control registers through the AXI Lite interface
- DMA reads and writes data to the DDR RAM using high performance (200 MB/s) AXI stream buses without bothering the processing system
- The DMA also reads and writes data to the accelerator component using similar AXI stream buses
- When tx and rx transmits are ready, the DMA sends an interrupt to the processing unit to inform that the task is finished. Therefore the ARM does not need to poll to check when the transfer is finished.



The implemented HW

Vaasan yliopisto
UNIVERSITY OF VAASA



Memory map

Vaasan yliopisto
UNIVERSITY OF VAASA

- The base address of the DMA control register is 0x4040_0000
- AXI-DMA has its own memory space in it's own 64 bit buses

Diagram | Address Editor

Cell	Slave Interface	Slave Segment	Offset Address	Range	High Address
processing_system7_0					
Data (32 address bits : 0x40000000 [1G])					
axi_dma_0	S_AXI_LITE	Reg	0x4040_0000	64K	0x4040_FFFF
axi_gpio_0	S_AXI	Reg	0x4120_0000	64K	0x4120_FFFF
axi_dma_0					
Data_SG (64 address bits : 16E)					
processing_system7_0	S_AXI_HP0	HP0_DDR_LOWOCM	0x0000_0000_0000_0000	1G	0x0000_0000_3FFF_FFFF
Data_MM2S (64 address bits : 16E)					
processing_system7_0	S_AXI_HP0	HP0_DDR_LOWOCM	0x0000_0000_0000_0000	1G	0x0000_0000_3FFF_FFFF
Data_S2MM (64 address bits : 16E)					
processing_system7_0	S_AXI_HP0	HP0_DDR_LOWOCM	0x0000_0000_0000_0000	1G	0x0000_0000_3FFF_FFFF

Device tree entry for axi dma and proxy

Vaasan yliopisto
UNIVERSITY OF VAASA

```
dma@40400000 {
        #dma-cells = <0x1>;
        clock-names = "s_axi_lite_aclk", "m_axi_sg_aclk", "m_axi_mm2s_aclk", "m_axi_s2mm_aclk"
        clocks = <0x1 0xf 0x1 0xf 0x1 0xf 0x1 0xf>;
        compatible = "xlnx,axi-dma-7.1", "xlnx,axi-dma-1.00.a";
        interrupt-names = "mm2s_introut", "s2mm_introut";
        interrupt-parent = <0x4>;
        interrupts = <0x0 0x1d 0x4 0x0 0x1e 0x4>;
        reg = <0x40400000 0x10000>;
        xlnx,addrwidth = <0x20>;
        xlnx,include-sg;
        xlnx,sg-include-stscntrl-strm;
        xlnx,sg-length-width = <0x1a>;
        phandle = <0x7>;

        dma-channel@40400000 {
                compatible = "xlnx,axi-dma-mm2s-channel";
                dma-channels = <0x1>;
                interrupts = <0x0 0x1d 0x4>;
                xlnx,datawidth = <0x20>;
                xlnx,device-id = <0x0>;
        };

        dma-channel@40400030 {
                compatible = "xlnx,axi-dma-s2mm-channel";
                dma-channels = <0x1>;
                interrupts = <0x0 0x1e 0x4>;
                xlnx,datawidth = <0x20>;
                xlnx,device-id = <0x0>;
        };
};
```

Write

Read

```
dma_proxy {
        compatible = "xlnx,dma_proxy";
        dmas = <0x7 0x0 0x7 0x1>;
        dma-names = "dma_proxy_tx", "dma_proxy_rx";
};
```

/dev/dma_proxy_tx
/dev/dma_proxy_rx