

# How to do this project right?

a.k.a. my things TODO wish list

**Elton Shih**

School of Computer Science and Engineering  
UNSW Sydney

COMP3601 23T3



**UNSW**  
SYDNEY

# Self Introduction



UNSW  
SYDNEY

## Elton Shih

- Research Engineer @Audinate
- Casual Academic @UNSW CSE (COMP1521, COMP2121/DESN2000, COMP3601)
- B.Eng, Computer Engineering, 2021

## COMP3601 Involvement

- 20T3 – Student (IR remote decoder box)
- 21T3 – Tutor (LWE crypto accelerator with approx. multipliers....)
- 22T3 – Helped redesign the course (COMP3601 x Audinate)



# Project Overview

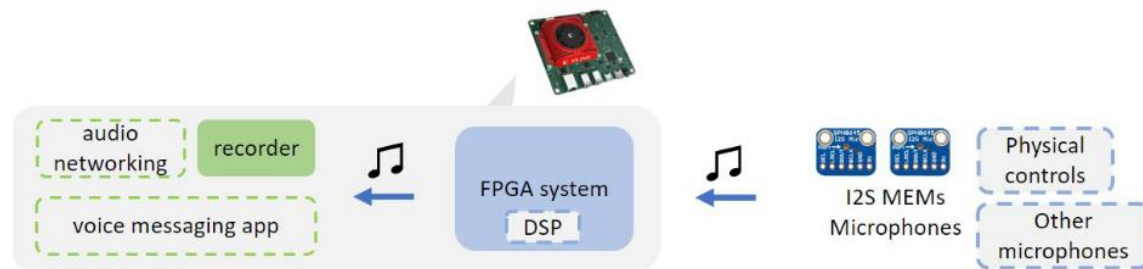
## Audio Pipeline Project

→ A “full-stack” project involving *electronics* (a bit), *digital systems design*, and *software*

Task: *Given a MEMs mic, a Kria board, and some starter code, get audio passing through to software land and go nuts on the extension*

### Milestone 2

Basic requirements: single MEMs mic -> PL -> PS (recorder)

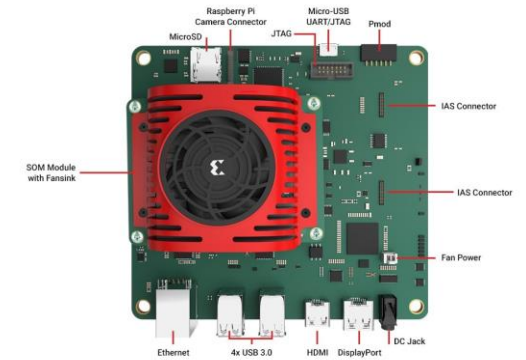


### Milestone 4

Application: “go nuts and show us what you got!”

**This is where you can get creative!**  
But I’m not here to tell you what you can do..  
→ technical side, what are things that can be done better

# My technical things TODO wish list



## Backstory (22T3)

- I spent a few weeks putting together the materials for this course
- I had a few wish lists when I did the course as a student and TA' in 21T3
  1. Replace the **Nexys A7** board (ideally with a SoC with on-chip FPGA)
  2. Hands-on development (21T3 was online, and the project was done purely on simulation)
  3. Students can go into COMP4601 with some handy skills learned in COMP3601
- What changed in last year's offering (22T3)
  1. Use Kria AI Starter Kits (Xilinx Zynq U+!)
  2. Back to hands-on development
  3. COMP4601 moving from using Zedboards to Kria
- Needed to provide some sort of boilerplate code due to the length of a term

*Unfortunately, it lacks PMOD pins and interfaces for electronic components.. (and the Kria robotics kit was yet to be introduced)*

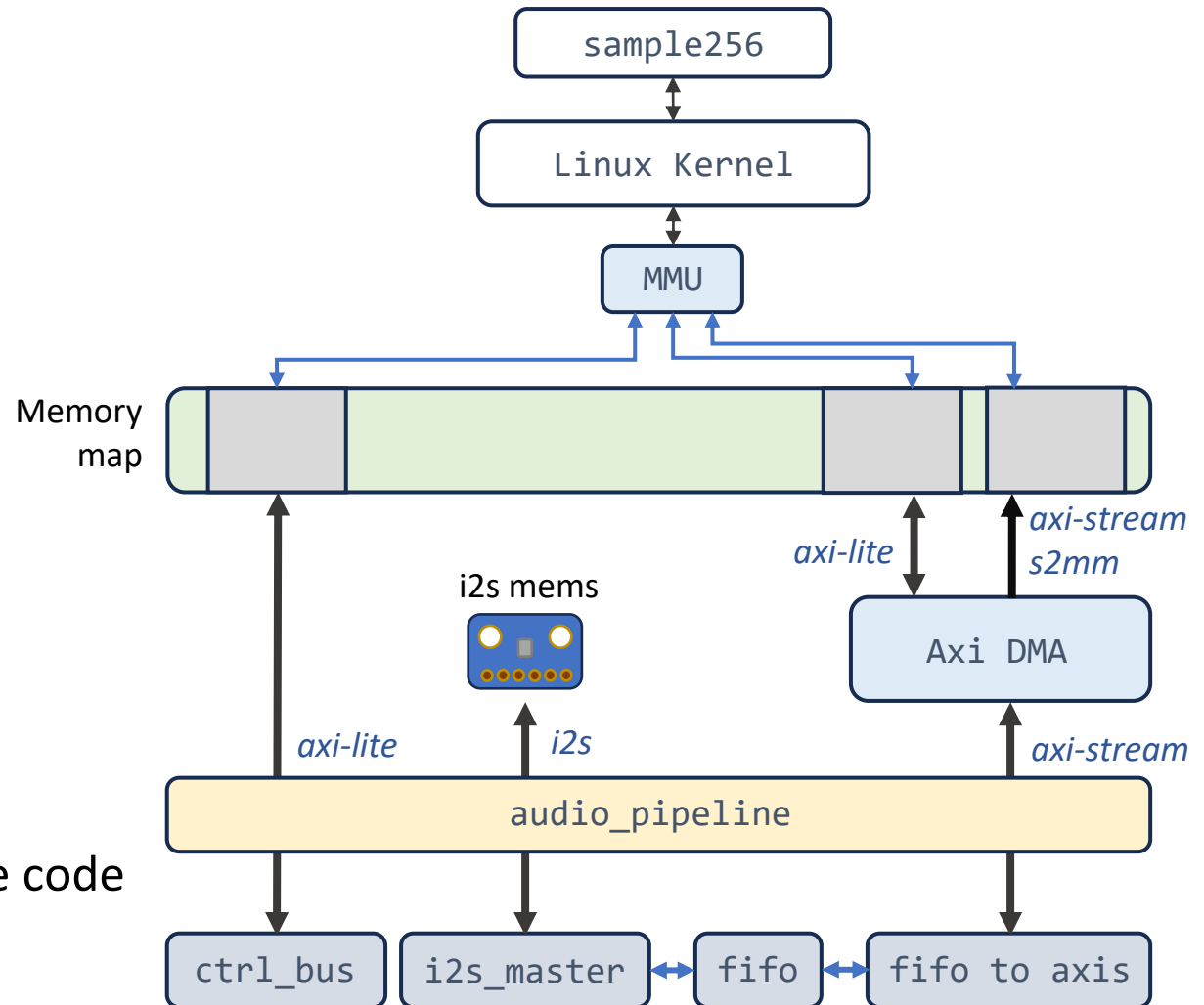
# My technical things TODO wish list (cont.)

## Boilerplate code (what's been provided to you)

- HDL: audio\_pipeline, i2s\_master, ctrl\_bus, fifo..
- SW: audio\_i2s, axi\_dma, main
- Bin: sample256, ctrl\_bus\_test
- Misc: OS image, SDK

## My TODO (what can be improved)

- Kernel space device driver for AXI DMA
- (or) at least register the buffer memory to the kernel
- Verify AXI stream transfers!
- You may have noticed, there are plenty of bugs in the code  
→ your chance to do better and do it right!



# AXI DMA driver

There are three AXI protocol flavours: *AXI-full*, *AXI-lite*, and *AXI-stream*

- **AXI-full** → high performance memory mapped data and address interface
- **AXI-lite** → similar to AXI-full but without burst capability (often used as the mem interface for our hardware designs in Xilinx land!)
- **AXI-stream** → Point-to-point (M2S) protocol for transferring data (higher rate). Usually used for moving a stream of data (e.g. video, audio, ethernet)

**AXI DMA** is used to transfer AXI Stream data between the FPGA and DDR memory.

*Instead of using CPU time to move data from FPGA to your memory, we use DMA's to help us move it -> CPU get to do something else..*

Table 2-3: AXI DMA Throughput Numbers<sup>(1)</sup>

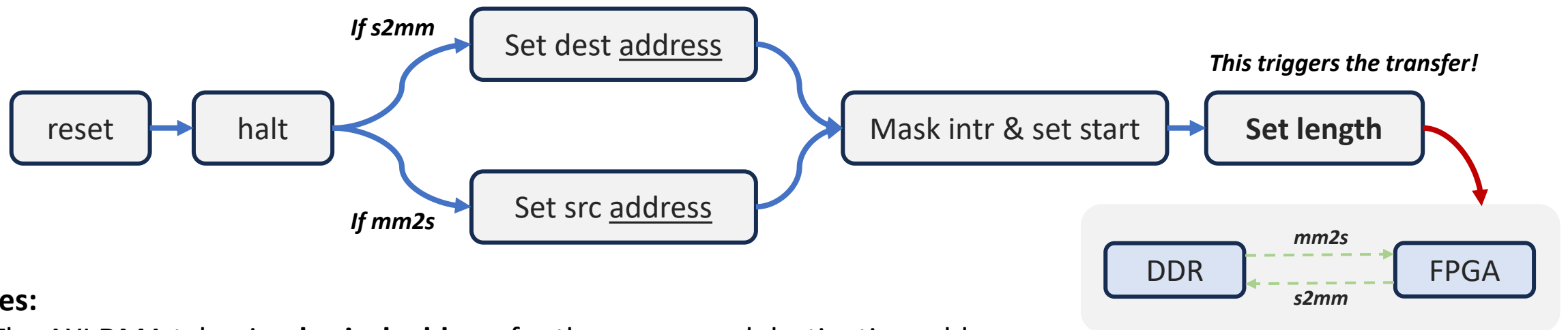
*pg021\_axi\_dma.pdf*

Channel	Clock Frequency (MHz)	Bytes Transferred	Total Throughput (MB/s)	Percent of Theoretical
MM2S <sup>(2)</sup>	100	10,000	399.04	99.76
S2MM <sup>(3)</sup>	100	10,000	298.59	74.64

**Exercise:** create a project (hw+sw) that measures the transfer throughput of AXI Stream and AXI Lite.

# AXI DMA driver (cont.)

Example flow of driving AXI DMA to perform transfers (s2mm or mm2s):



## Notes:

- The AXI DMA takes in **physical address** for the source and destination addresses
- On transfer completion, the `IOC_Irq` signal on the corresponding (S2MM or MM2S) status register should be fired  
(Interrupt on Complete)

## The provided `axi_dma` user space driver (`axi_dma.c/.h`):

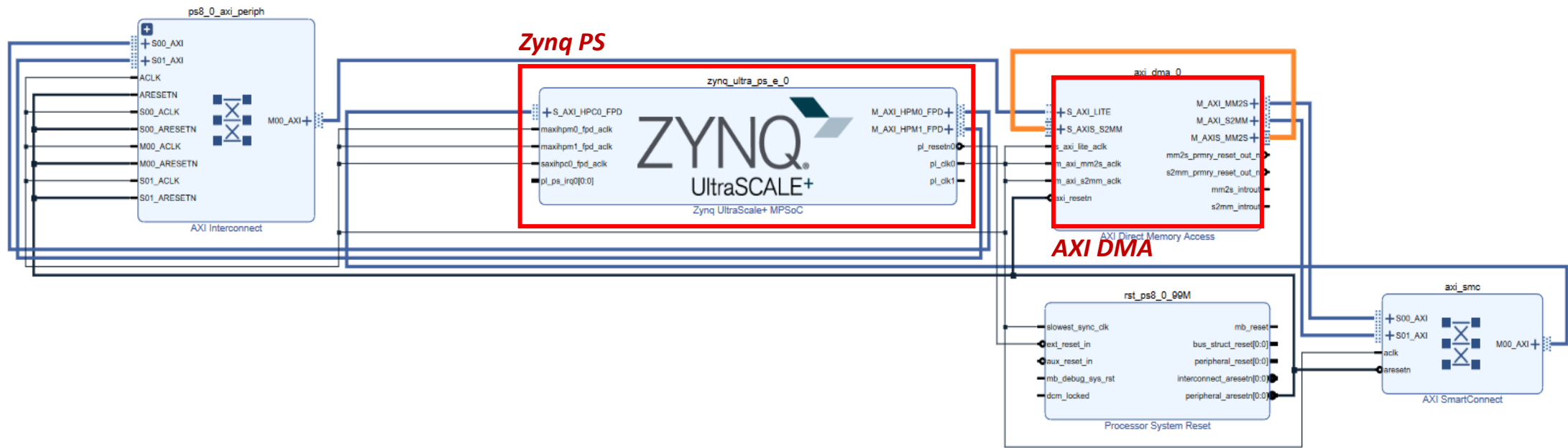
- Memory maps the register map of AXI DMA controller
- Memory maps a chosen physical address space chunk (within the DDR range)
- Provides the physical address to the AXI DMA controller for the src and dest addresses
- Busy waits for `IOC_Irq` signal being flagged! (how is this possible :0)

*Exercise: try connecting the `s2mm_introut` to `pl_ps_irq`, synthesize, and run. What happens to the transfer's busy wait?*

# AXI Stream transfers in Petalinux vs RTOS/baremetal

## The Vivado design flow is the same

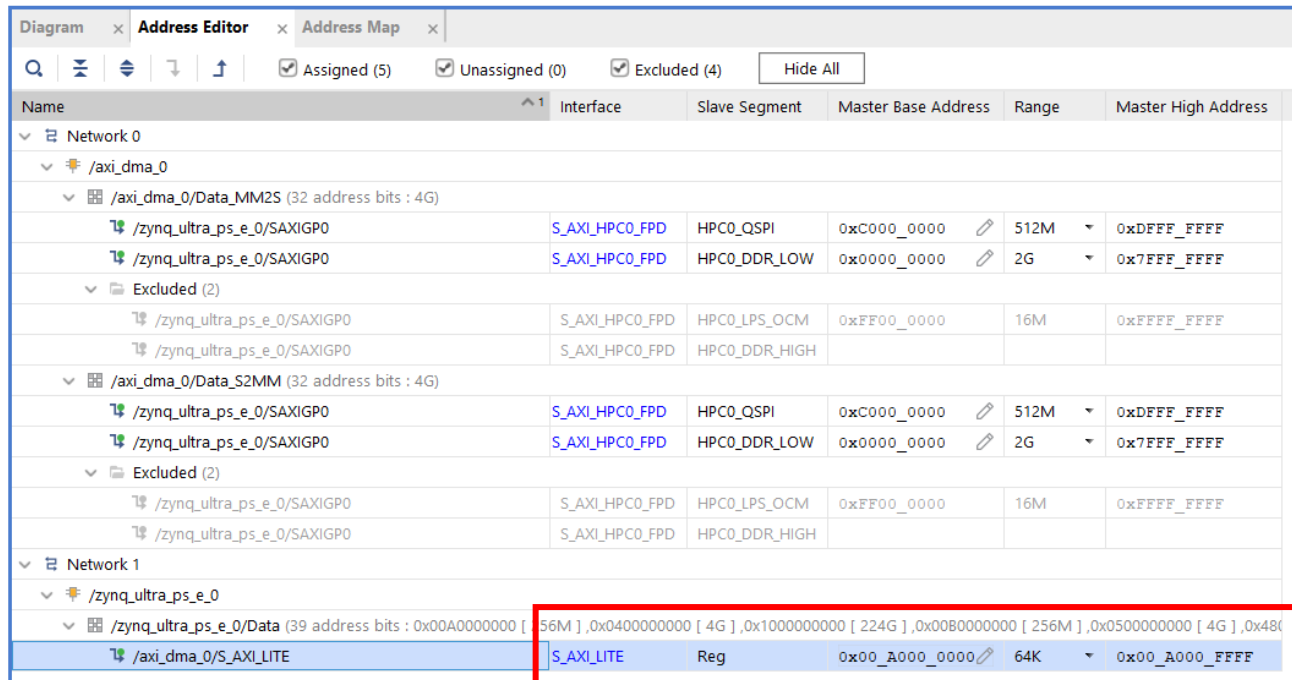
- You design your digital system (that speaks axis), create a system block design
- Add the Zynq IP block, add your design, add an AXI DMA block, wire them up!



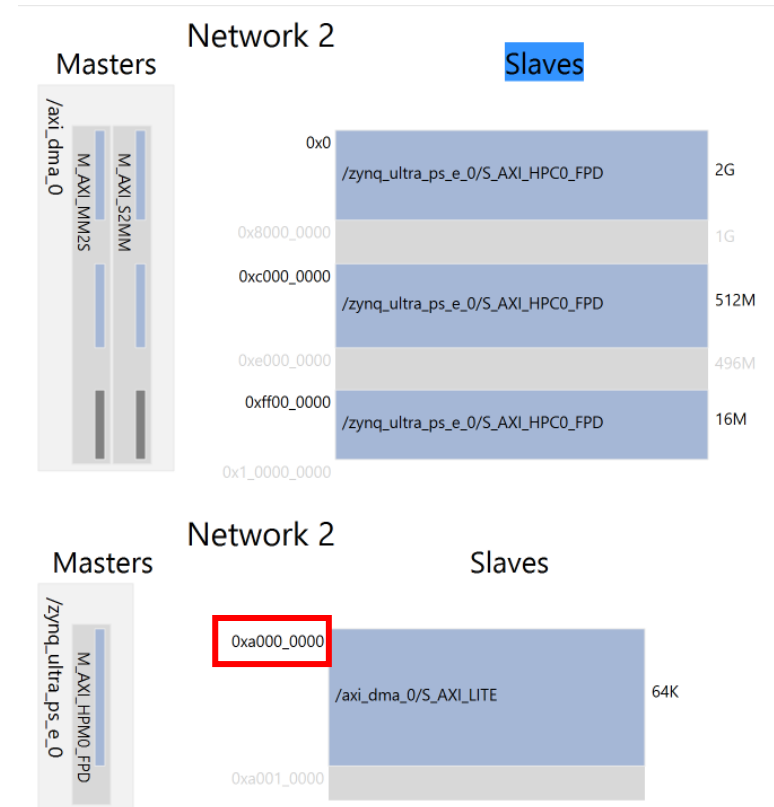


# AXI Stream transfers in Petalinux vs RTOS/baremetal

- Vivado automatically helps you arrange your register-mapped devices to the memory map of the PS
- axi-lite interfaces connected to the HP ports will be mapped



Name	Interface	Slave Segment	Master Base Address	Range	Master High Address
<b>Network 0</b>					
/axi_dma_0					
/axi_dma_0/Data_MM2S (32 address bits : 4G)					
/zynq_ultra_ps_e_0/SAXIGP0	S_AXI_HPC0_FPD	HPC0_QSPI	0xC000_0000	512M	0xDFFF_FFFF
/zynq_ultra_ps_e_0/SAXIGP0	S_AXI_HPC0_FPD	HPC0_DDR_LOW	0x0000_0000	2G	0x7FFF_FFFF
Excluded (2)					
/zynq_ultra_ps_e_0/SAXIGP0	S_AXI_HPC0_FPD	HPC0_LPS_OCM	0xFF00_0000	16M	0xFFFF_FFFF
/zynq_ultra_ps_e_0/SAXIGP0	S_AXI_HPC0_FPD	HPC0_DDR_HIGH			
/axi_dma_0/Data_S2MM (32 address bits : 4G)					
/zynq_ultra_ps_e_0/SAXIGP0	S_AXI_HPC0_FPD	HPC0_QSPI	0xC000_0000	512M	0xDFFF_FFFF
/zynq_ultra_ps_e_0/SAXIGP0	S_AXI_HPC0_FPD	HPC0_DDR_LOW	0x0000_0000	2G	0x7FFF_FFFF
Excluded (2)					
/zynq_ultra_ps_e_0/SAXIGP0	S_AXI_HPC0_FPD	HPC0_LPS_OCM	0xFF00_0000	16M	0xFFFF_FFFF
/zynq_ultra_ps_e_0/SAXIGP0	S_AXI_HPC0_FPD	HPC0_DDR_HIGH			
<b>Network 1</b>					
/zynq_ultra_ps_e_0					
/zynq_ultra_ps_e_0/Data (39 address bits : 0x00A0000000 [ 56M ] , 0x0400000000 [ 4G ] , 0x1000000000 [ 224G ] , 0x0080000000 [ 256M ] , 0x0500000000 [ 4G ] , 0x4800000000 [ 256M ] )					
/axi_dma_0/S_AXI_LITE	S_AXI_LITE	Reg	0x00_A000_0000	64K	0x00_A000_FFFF



# AXI Stream transfers in Petalinux vs RTOS/baremetal

## Baremetal and RTOS

- Firmware/application code runs in physical address space (no virtual address translation!)
- **If I want to access 0x40400000**, AXI DMA's interface, which is the mm2s control register
  - Baremetal and RTOS programs can directly use the 0x40400000 address
- **If I want to use a section of DDR directly as the transfer buffer**
  - Baremetal and RTOS programs can directly use and provide the physical address to the src and dest address register of the AXI DMA

Table 2-8: Direct Register Mode Register Address Map

Address Space Offset <sup>(1)</sup>	Name	Description
00h	MM2S_DMOCR	MM2S DMA Control register
04h	MM2S_DMASR	MM2S DMA Status register
08h – 14h	Reserved	N/A

## Petalinux

- Each user program process has its own virtual address space
- Process pages have virtual addresses and are translated into physical address
- **If I want to access 0x40400000**, AXI DMA's interface, which is the mm2s control register
  - User space programs will have to memory map the device (0x40400000) to the processes virtual address space
- **If I want to use a section of DDR directly as the transfer buffer**
  - User space programs will have to memory map the buffers' physical address to the virtual address space before use
  - But still require to provide the physical address of the buffer to the src and dest address register

# AXI Stream transfers in Petalinux vs RTOS/baremetal

## Petalinux

- However, this may be problematic!
- Your user space process decides to make use of a chunk of memory in DDR, but the kernel processes don't know..
- what happens if the scheduler pre-empts your process and a random process decides to put something in your DDR buffer range?

How to solve this? A few ways...

1. Reserve a memory chunk in the device tree (system-top.dts)
2. Use the CMA reserved memory..(?)



# Reserving memory I

## 1. system-top.dts node for reserved memory

```
reserved-memory {
    #address-cells = <2>;
    #size-cells = <2>;
    ranges;

    reserved: buffer@0 {
        compatible = "shared-dma-pool";
        no-map;
        reg = <0x0 0x70000000 0x0 0x10000000>;
    };
};

reserved-driven@0 {
    compatible = "xlnx,reserved-memory";
    memory-region = <&reserved>;
};
```

## 2. Device driver using DMA API on the reserved memory (not CMA)

```
/* Get reserved memory region from Device-tree */
np = of_parse_phandle(dev->of_node, "memory-region", 0);
if (!np) {
    dev_err(dev, "No %s specified\n", "memory-region");
    goto error1;
}

rc = of_address_to_resource(np, 0, &r);
if (rc) {
    dev_err(dev, "No memory address assigned to the region\n");
    goto error1;
}

lp->paddr = r.start;
lp->vaddr = memremap(r.start, resource_size(&r), MEMREMAP_WB);
dev_info(dev, "Allocated reserved memory, vaddr: 0x%011X, paddr: 0x%011X\n", (u64)lp->vaddr, lp->paddr);
```

## 3. Iomem showing our region being excluded from kernel ucommon usage

```
root@plnx aarch64:~# cat /proc/iomem
00000000-6fffffff : System RAM
00080000-00b37fff : Kernel code
011c9000-012b8fff : Kernel data
```

## 4. Kernel bootlog after loading device driver

```
[ 126.191774] reserved-memory reserved-driver@0: Device Tree Probing
[ 126.198595] reserved-memory reserved-driver@0: Allocated reserved memory, vaddr: 0xFFFFFFFF802000000, paddr: 0x70000000
```

# Reserving memory II

## 1. system-top.dts node for reserved memory (same as I)

```
reserved-memory {
    #address-cells = <2>;
    #size-cells = <2>;
    ranges;

    reserved: buffer@0 {
        compatible = "shared-dma-pool";
        no-map;
        reg = <0x0 0x70000000 0x0 0x10000000>;
    };
};

reserved-drivers@0 {
    compatible = "xlnx,reserved-memory";
    memory-region = <&reserved>;
};
```

## 2. Device driver using DMA API on the reserved memory (not CMA)

```
/* Initialize reserved memory resources */
rc = of_reserved_mem_device_init(dev);
if(rc) {
    dev_err(dev, "Could not get reserved memory\n");
    goto error1;
}

/* Allocate memory */
dma_set_coherent_mask(dev, 0xFFFFFFFF);
lp->vaddr = dma_alloc_coherent(dev, ALLOC_SIZE, &lp->paddr, GFP_KERNEL);
dev_info(dev, "Allocated coherent memory, vaddr: 0x%011X, paddr: 0x%011X\n", (u64)lp->vaddr, lp->paddr);
```

## 3. Kernel Bootlog

```
[ 0.000000] Reserved memory: created DMA memory pool at 0x0000000070000000, size 256 MiB
[ 0.000000] Reserved memory: initialized node buffer@0, compatible id shared-dma-pool
[ 0.000000] cma: Reserved 128 MiB at 0x0000000068000000
```

## 4. Kernel bootlog after loading device driver

```
root@plnx_aarch64:~# insmod /lib/modules/4.6.0-xilinx/extra/reserved-memory.ko
[ 80.745166] reserved-memory reserved-drivers@0: Device Tree Probing
[ 80.750183] reserved-memory reserved-drivers@0: assigned reserved memory node buffer@0
[ 81.220878] reserved-memory reserved-drivers@0: Allocated coherent memory, vaddr: 0xFFFFFFFF8020000000, paddr: 0x70000000
```

# How far do we need to go?

Can we still use our userspace axi\_dma driver?

- Yes!

```
reserved-memory {
    #address-cells = <2>;
    #size-cells = <2>;
    ranges;

    reserved: buffer@0 {
        compatible = "shared-dma-pool";
        no-map;
        reg = <0x0 0x70000000 0x0 0x10000000>;
    };

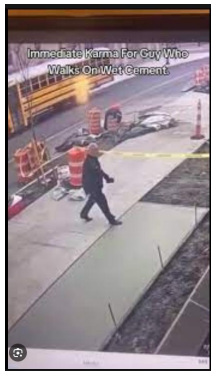
    reserved-driver@0 {
        compatible = "xlnx,reserved-memory";
        memory-region = <&reserved>;
    };
};
```

```
[ 0.000000] Reserved memory: created DMA memory pool at 0x0000000070000000, size 256 MiB
[ 0.000000] Reserved memory: initialized node buffer@0, compatible id shared-dma-pool
[ 0.000000] cma: Reserved 128 MiB at 0x0000000068000000
```

**Q: what about the CMA preserved memory?**

Sure, but can we be sure that other device drivers that allocate memory from CMA won't get allocated overlapping memory? Not really..

But we could use the CMA pool if we (axi dma driver) are in kernel land



# I'm not a LKM expert but...

Here is how you can create your own Linux module for Kria:

- Prereq: **setup the petalinux tool** or have the required files setup on your OS image
  - Install petalinux, source the settings.sh
- Building through the petalinux tool
  - `petalinux-create --type project -s xilinx-k26-starterkit-v2021.1-final.bsp --name 3601_plx`
  - `petalinux-config`
  - `petalinux-create -t modules --name mymodule --enable`
  - `petalinux-build -c mymodule` → *only builds the module (after the first build..)!*
- Artefacts are located in `<TMPDIR>/work/<MACHINE_NAME>-xilinx-linux/mymodule/1.0-r0/`

**Warning:** Petalinux isn't really a "straightforward and lightweight tool", generally okay to use but you might hit some frustrations

# Loading your module

- lsmod
- insmod <kernel module object>
- rmmod <kernel module object>
- modprobe <-r|I|D>

## 3. Loading and dmesg (after)

```
root@xilinx-k26-starterkit-2021_1:~/experiment# insmod hello.ko
root@xilinx-k26-starterkit-2021_1:~/experiment# dmesg | tail
[ 11.346330] usb 1-1.5: new high-speed USB device number 5 using xhci-hcd
[ 11.451016] usb 1-1.5: New USB device found, idVendor=0424, idProduct=2740, bcdDevice= 2.00
[ 11.451025] usb 1-1.5: New USB device strings: Mfr=1, Product=2, SerialNumber=0
[ 11.451030] usb 1-1.5: Product: Hub Controller
[ 11.451034] usb 1-1.5: Manufacturer: Microchip Tech
[ 23.062580] random: crng init done
[ 23.062590] random: 2 urandom warning(s) missed due to ratelimiting
[ 29.080141] process 'docker/tmp/qemu-check467939191/check' started with executable stack
[ 875.424092] <1>Hello module world.
[ 875.424103] <1>Module parameters were (0xdeadbeef) and "default"
root@xilinx-k26-starterkit-2021_1:~/experiment#
```

## 1. dmesg (before)

```
root@xilinx-k26-starterkit-2021_1:~/experiment# dmesg | tail
[ 11.136216] hub 2-1:1.0: USB hub found
[ 11.136305] hub 2-1:1.0: 4 ports detected
[ 11.346330] usb 1-1.5: new high-speed USB device number 5 using xhci-hcd
[ 11.451016] usb 1-1.5: New USB device found, idVendor=0424, idProduct=2740, bcdDevice= 2.00
[ 11.451025] usb 1-1.5: New USB device strings: Mfr=1, Product=2, SerialNumber=0
[ 11.451030] usb 1-1.5: Product: Hub Controller
[ 11.451034] usb 1-1.5: Manufacturer: Microchip Tech
[ 23.062580] random: crng init done
[ 23.062590] random: 2 urandom warning(s) missed due to ratelimiting
[ 29.080141] process 'docker/tmp/qemu-check467939191/check' started with executable stack
root@xilinx-k26-starterkit-2021_1:~/experiment#
```

## 2. lsmod (before)

```
root@xilinx-k26-starterkit-2021_1:~/experiment# lsmod
Module                Size  Used by
xt_contrack           16384  1
xt_MASQUERADE         16384  1
xt_addrtype           16384  2
iptable_filter        16384  1
iptable_nat           16384  1
nf_nat                36864  2 iptable_nat,xt_MASQUERADE
usb5744               16384  0
dmaproxy              16384  0
mali                  233472  0
uio_pdrv_genirq       16384  0
root@xilinx-k26-starterkit-2021_1:~/experiment#
```

## 3. lsmod (after)

```
root@xilinx-k26-starterkit-2021_1:~/experiment# lsmod
Module                Size  Used by
hello                 16384  0
xt_contrack           16384  1
xt_MASQUERADE         16384  1
xt_addrtype           16384  2
iptable_filter        16384  1
iptable_nat           16384  1
nf_nat                36864  2 iptable_nat,xt_MASQUERADE
usb5744               16384  0
dmaproxy              16384  0
mali                  233472  0
uio_pdrv_genirq       16384  0
root@xilinx-k26-starterkit-2021_1:~/experiment#
```



# Go nuts with your project (I) - If you have time...

And if you are really interested in the software system land of work

And if you are confident that you can finish it properly within the rest of the term

- Try implementing the device driver as kernel modules instead of user space driver
- Even more nuts, making your microphone work with the ALSA ecosystem..

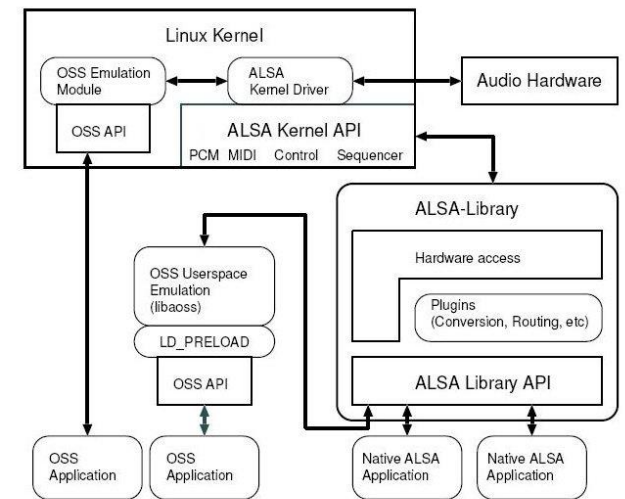


Figure 1: Basic Structure and Flow of ALSA System

[https://en.opensuse.org/SDB:Sound\\_concepts](https://en.opensuse.org/SDB:Sound_concepts)

# Readings

- <https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18841683/Linux+Reserved+Memory>
- <https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18842418/Linux+DMA+From+User+Space>
- <https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/1027702787/Linux+DMA+From+User+Space+2.0>
- [https://users.ece.utexas.edu/~mcdermot/arch/articles/Zynq/pg021\\_axi\\_dma.pdf](https://users.ece.utexas.edu/~mcdermot/arch/articles/Zynq/pg021_axi_dma.pdf)
- [https://docs.xilinx.com/v/u/en-US/ug761\\_axi\\_reference\\_guide](https://docs.xilinx.com/v/u/en-US/ug761_axi_reference_guide)
- <https://lauri.võsandi.com/hdl/zynq/xilinx-dma.html>
- <https://www.realdigital.org/doc/a9fee931f7a172423e1ba73f66ca4081>

# Non-busy-wait AXI Stream transfer

- Have you noticed a warning when wiring up your block design in Vivado?
  - Something related to the s2mm\_introut and mm2s\_introut?
- This was intended.
  - Without connecting, when transfers complete, the IOC\_irq remains flagged so we can do busy polling of this signal to know whether the program has been completed or not
  - But is busy-polling preferred?
- Hints on not doing busy-polling:
  - Connect the mm2s and s2mm introut signal to the Zynq device's interrupt port
  - Think about how interrupts work and how to write an interrupt handler!

# Brief intro to booting on the Kria

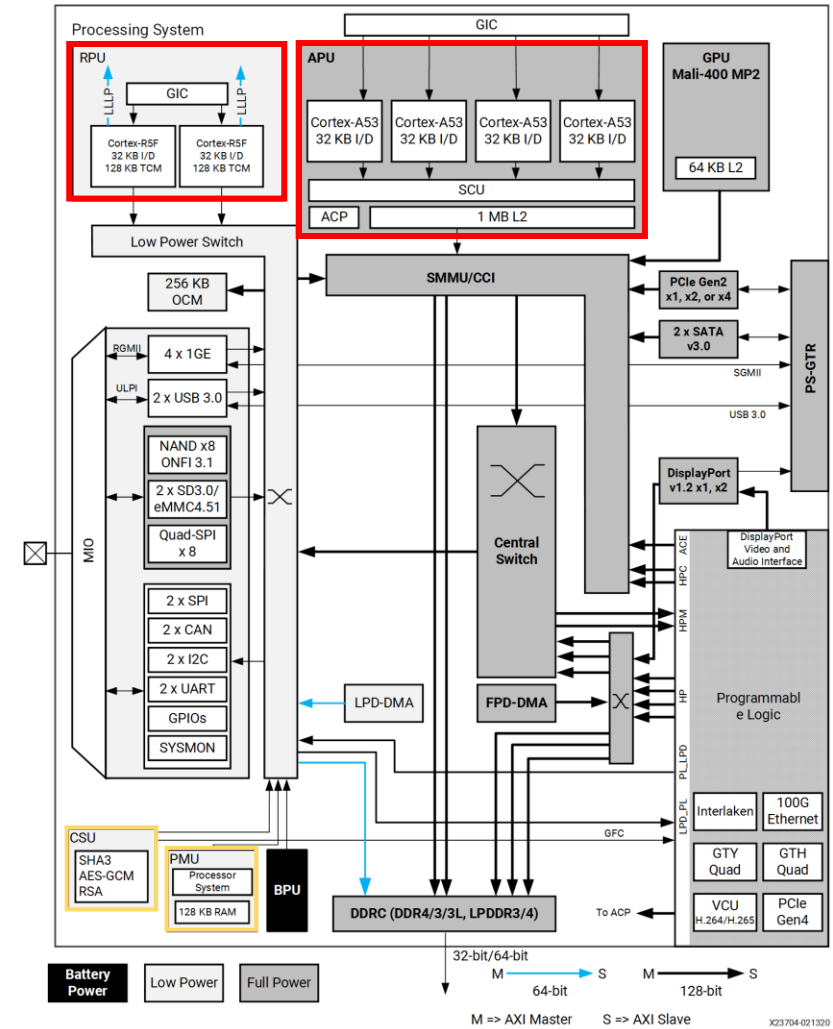
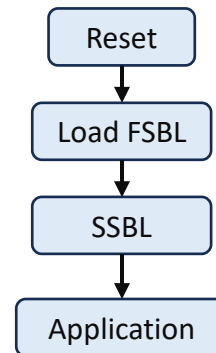
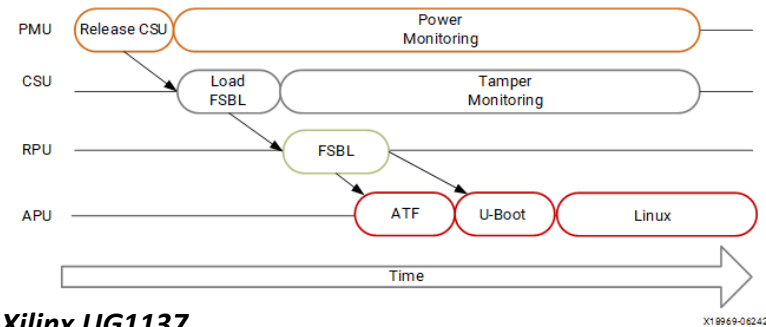
## What's on the Kria?

- ARM Cortex A53 x4 → **APU**
- ARM Cortex R5 x2 → **RPU**
- **PMU** → Platform Management Unit
- **CSU** → Configuration and Security Unit

## For software, you can choose to run Linux, an RTOS, or baremetal software

- Linux → PetaLinux based or Ubuntu on A53
- RTOS → FreeRTOS, Zephyr RTOS, or others on R5
- Baremetal -> direct drivers and libraries and running on R5

## Simplified boot process (Linux)



*Xilinx UG1137*

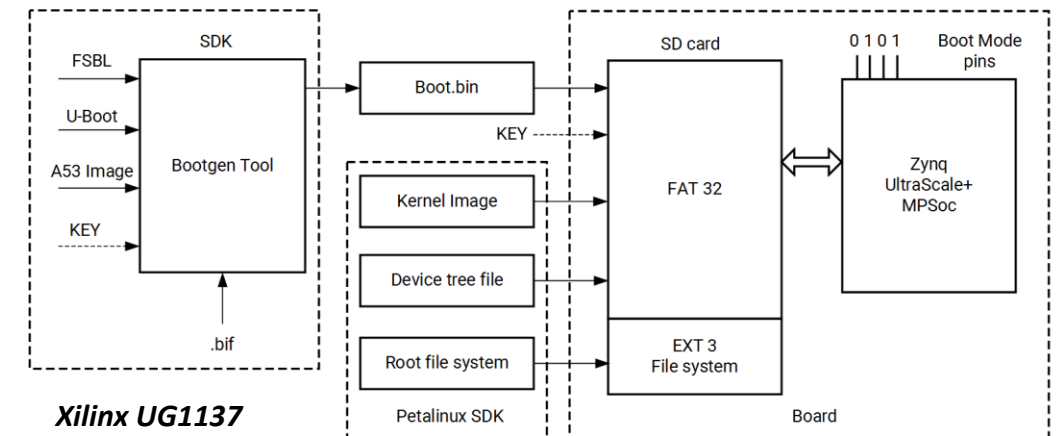
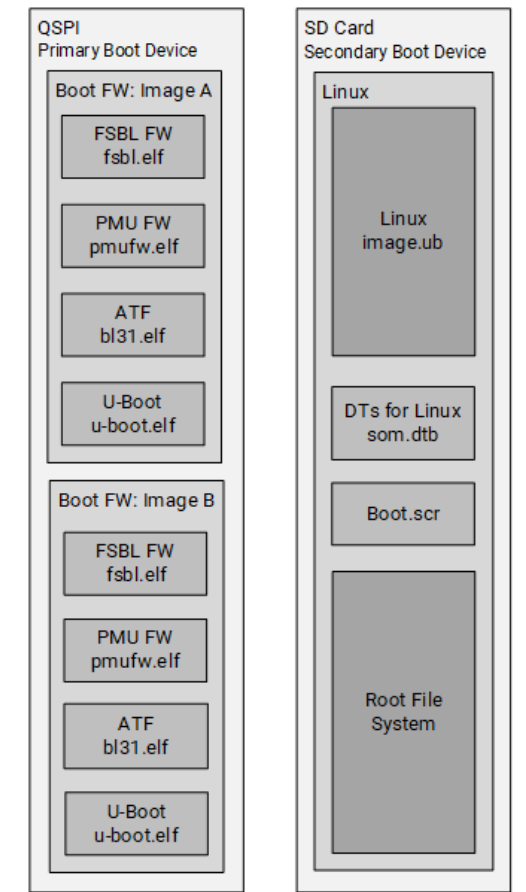
# Detailed view of booting via SD card

## Files you need to boot Linux (all prebuilt for you)

- Boot firmware (BOOT.BIN) → **FSBL**
  - First stage bootloader firmware (fsbl.elf)
  - PMU firmware (pmufw.elf)
  - ARM Trusted Firmware (bl31.elf)
  - Second stage BL (u-boot.elf)
- Linux → **SSBL** (u-boot and Linux) #wk3-lecture
  - System device tree blob (system.dtb or som.dtb)
  - Linux image (image.ub)
  - Boot script (boot.scr)
  - Root filesystem ("/", contains libraries, configs, binaries etc)

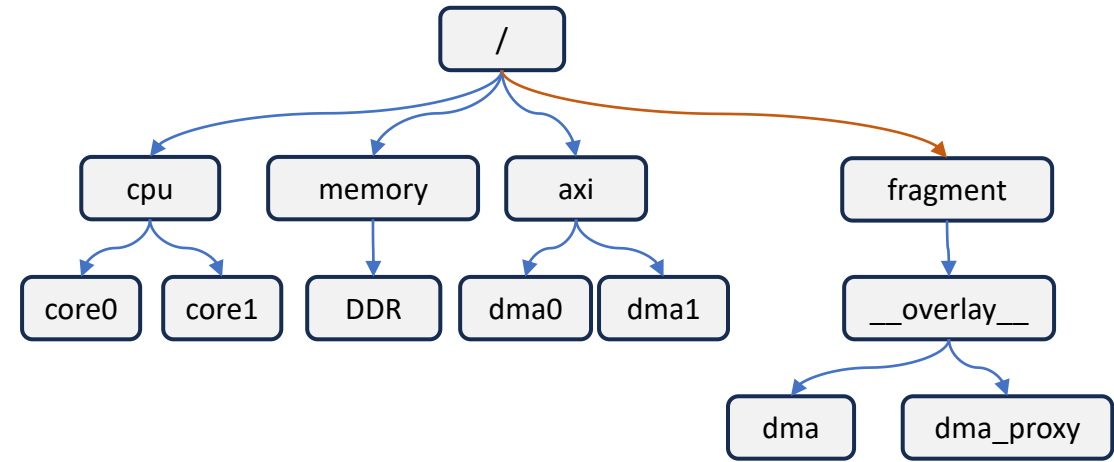
→ All pre-built using Petalinux (Yocto-based tool for image and sdk generation)

→ *Unless there is a very specific reason you need to rebuild the image, I suggest leaving it as it is..*



# Device Trees

- A way to describe hardware to the software system
- Kernel does not need to hard code details of the machine
- Introduced in Linux Kernel 2.6



```

/ {
    node@0 {
        a-string-property = "A string";
        a-string-list-property = "first string", "second string";
        a-byte-data-property = [0x01 0x23 0x34 0x56];

        child-node@0 {
            first-child-property;
            second-child-property = <1>;
            a-reference-to-something = <&node1>;
        };

        child-node@1 {
        };
    };

    node1: node@1 {
        an-empty-property;
        a-cell-property = <1 2 3 4>;

        child-node@0 {
        };
    };
};
  
```

Thomas Petazzoni, Device Tree for Dummies

**dts (device tree source)** and **dtsi (dts include file)** are in human-readable representations of Device Tree data.

**dtb (device tree binary/blob)** and **dtbo (dtb overlay)** are compiled dts files represented in binary format.

You can compile dts(i) files into dtb(o) and decompile dtb(o) into dts(i) with the **dtc (device tree compiler)** tool.

- system-top.dts** -> contains memory info, boot args and early console args
- pl.dtsi** -> contains mem mapped PL peripheral IP nodes
- zynqmp.dtsi** -> contains PS peripheral and CPU nodes
- zynqmp-clk-ccf.dtsi** -> contains clock info for peripheral IPs

# Device Trees (cont.)

Device trees provide huge flexibility for embedded work.

→ You can statically link it to the kernel, loaded by the bootloader, or during runtime!

Have you thought about how the xutil tool can swap out bitstream and the hardware modules in the PL and the system still recognizes new blobs? **Recall the DTS generation stage and the final artefacts you load in Kria.**

Vivado generates the “view of your hardware” from PS perspective (XSA → DTS → DTBO).

→ 

```
root@xilinx-k26-starterkit-2021_1:/lib/firmware/xilinx/accel_dbg# ls
accel_dbg.bit.bin  accel_dbg.dtbo  shell.json
```

- **<app>.bit.bin** is the headerless bitstream binary → gets loaded into the system
- **<app>.dtbo** is the device tree overlay → loaded onto the runtime device tree
- **shell.json** → describes the base shell configuration

# Readings

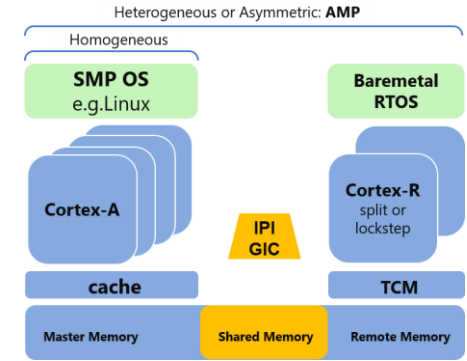
- <https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18842279/Build+Device+Tree+Blob>
- [https://elinux.org/Device Tree Usage](https://elinux.org/Device_Tree_Usage)
- <https://github.com/Xilinx/dfx-mgr>



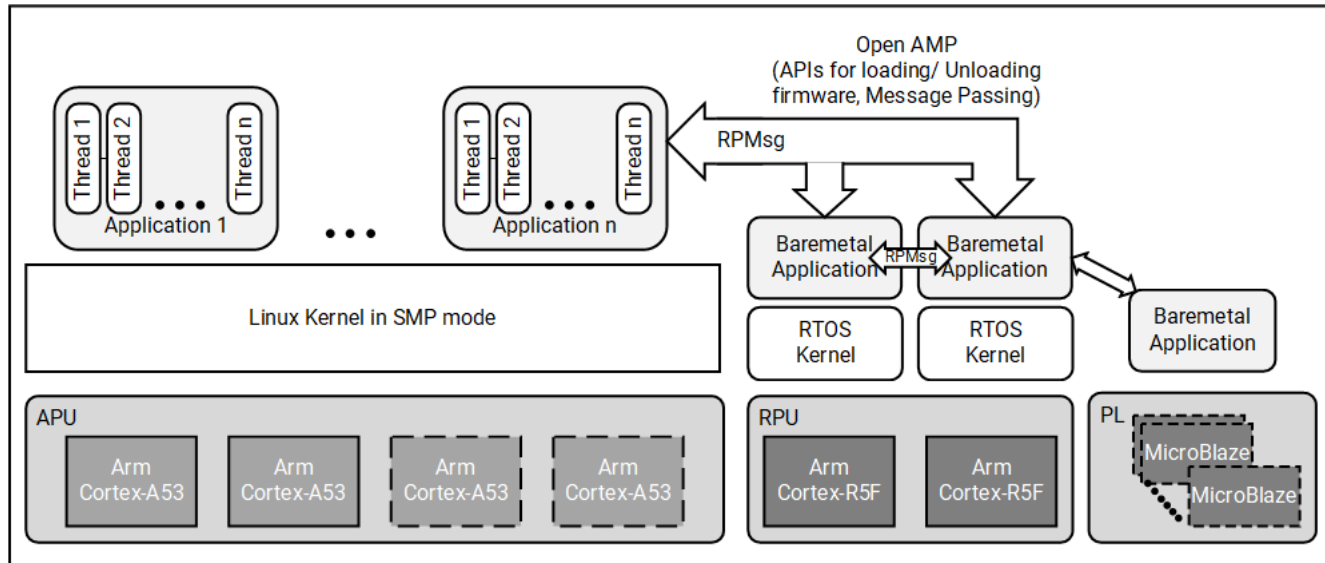
# Go nuts with your project (II) - If you have time...

And if you are interested in playing with the system

- Try running Petalinux on APU and RTOS/baremetal of your choice on RPU
- Have some sort of control interface managed by RPU for your application!

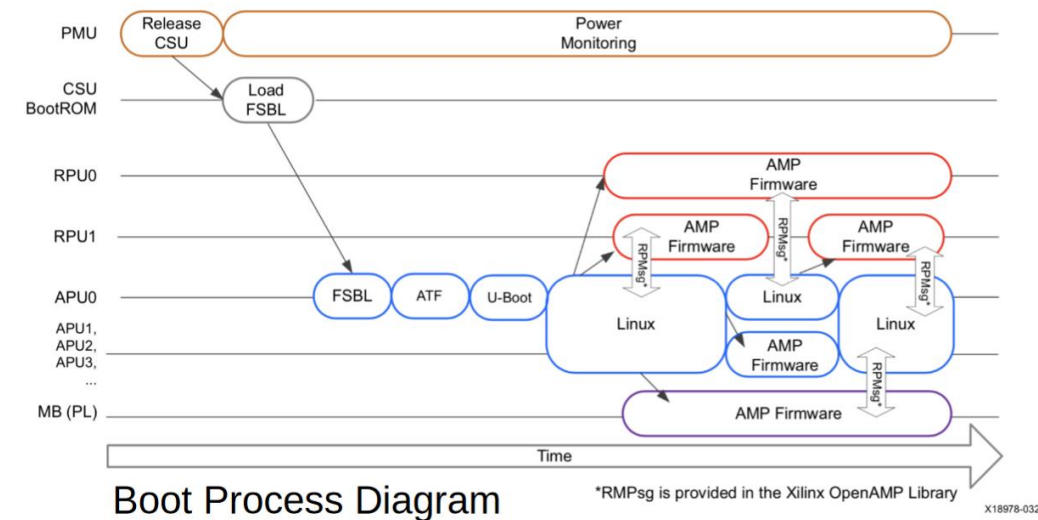


You may need OpenAMP (supported starting from 2022.1) (zephyr support as well..)



Xilinx UG1137

X14839-063017



Xilinx UG1137

# Readings

- [https://xilinx.github.io/kria-apps-docs/openamp/build/html/openamp\\_landing.html](https://xilinx.github.io/kria-apps-docs/openamp/build/html/openamp_landing.html)
- <https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18841718/OpenAMP>
- [https://docs.zephyrproject.org/latest/boards/arm/kv260\\_r5/doc/index.html](https://docs.zephyrproject.org/latest/boards/arm/kv260_r5/doc/index.html)

# Some misc software/app ideas..

- Audio over network?
  - Each group have two boards
  - Can you set up a VoIP system between the two boards with your audio system?
  - (not – record, send, playback but continuous data streams!)
  - Latency! Have a look at Dante if interested, but for those interested, I encourage you to develop your own system.
- Samples -> PS driver -> directly packetized and sent over the network to the other board.

# One final suggestion..

- Get the basic version completed before challenging yourself
- Know the system, do lots of googling, and try things yourself!
- Have fun!

# Questions...