6.S965
Digital Systems Laboratory II

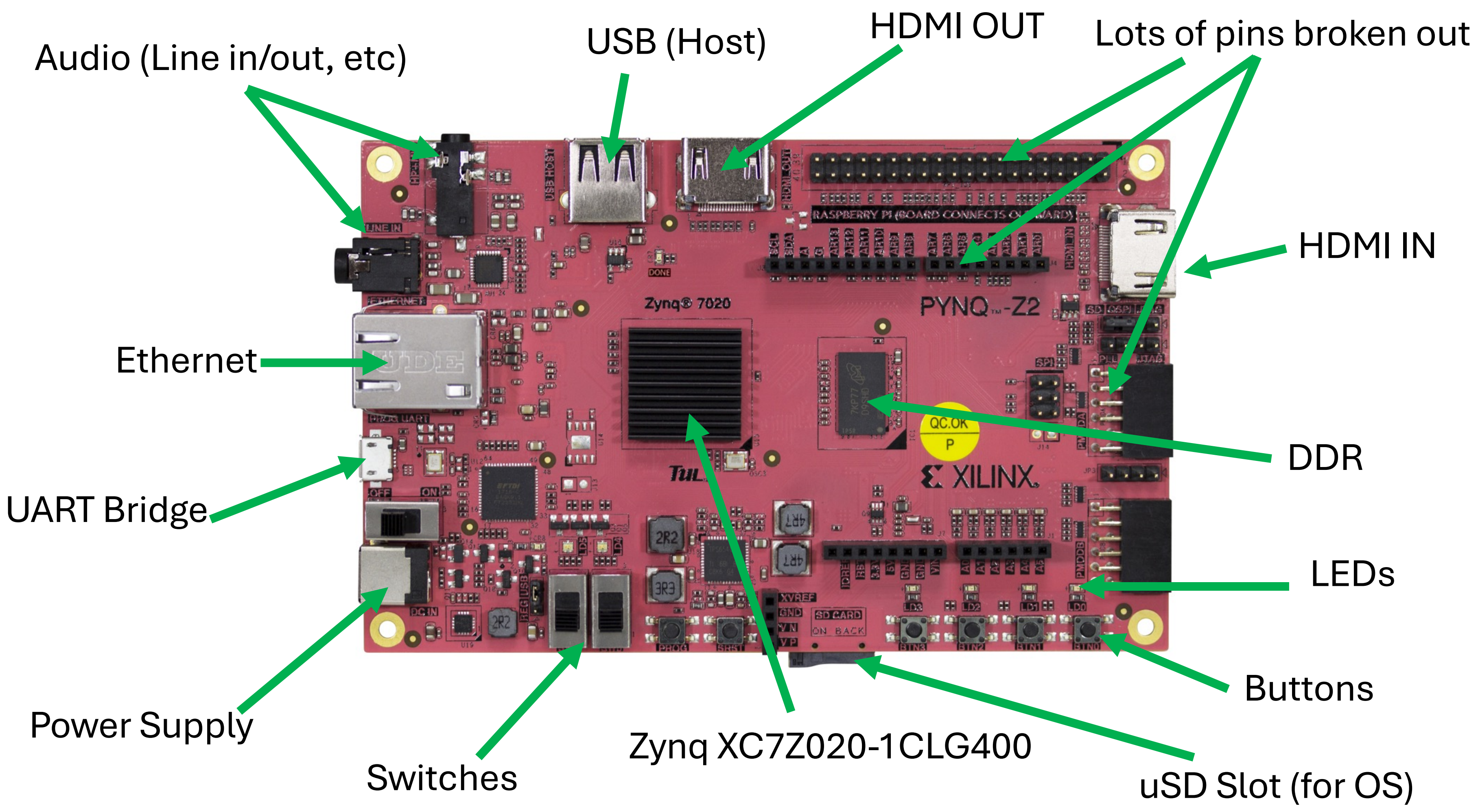
Lecture 3:

Zynq Architecture

Administrative

- Week 1’s stuff due Friday at 5pm
- Week 2’s stuff should be out at noon on Friday
- If you find yourself thinking, “I’m probably doing something stupid…” in the context of Vivado, the problem may not be you, it may be Vivado. Please ask for help

Some Stuff on the PYNQ Z2 Board

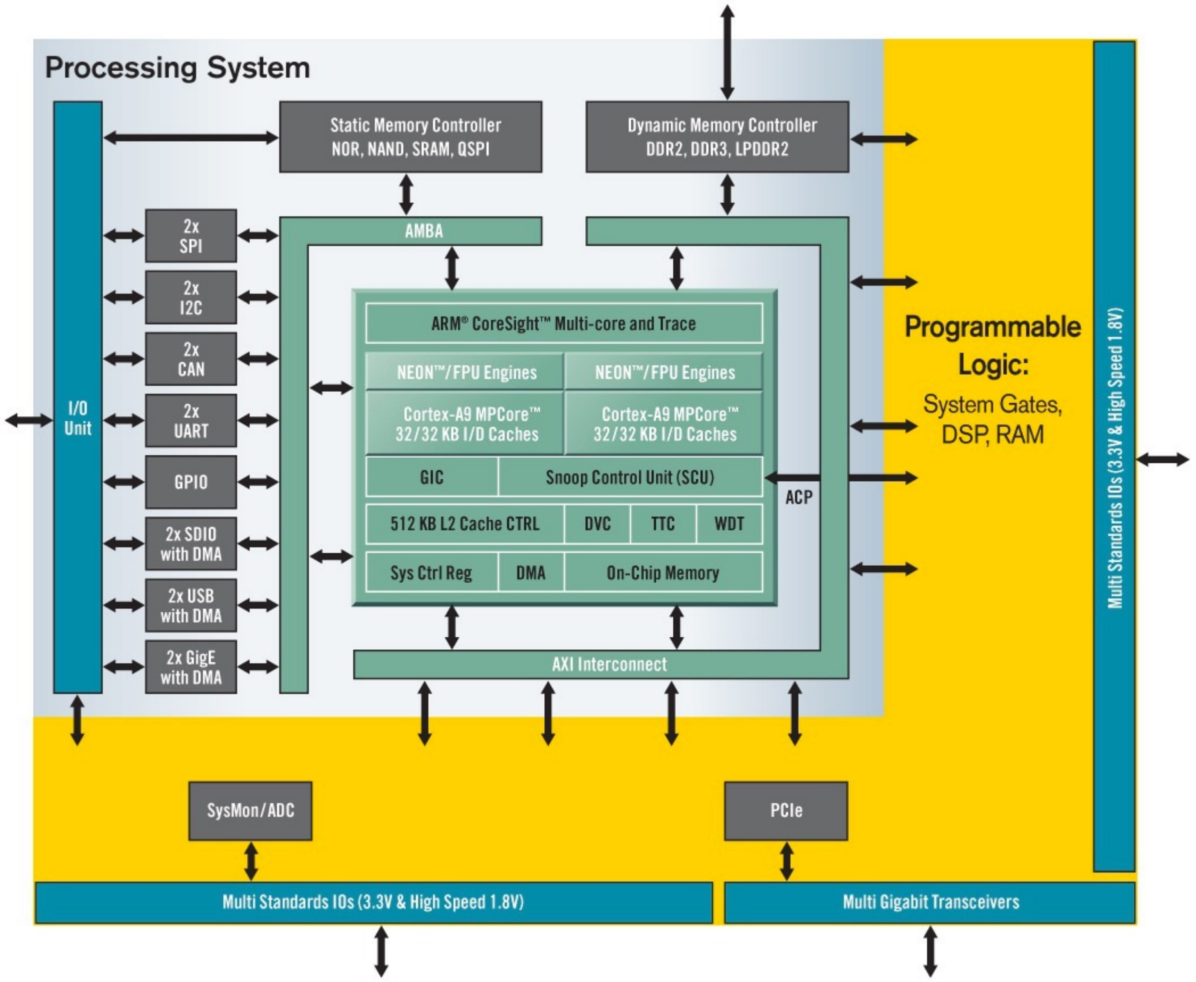


How Can You Work With it?

- The Zynq XC7Z020-1CLG400 has almost twice the amount of “classic” FPGA material as the Spartan 7 boards used in 6.205
  - 13,300 Logic Cells
  - 630 KByte of BRAM
  - 220 DSP slices
  - On-chip analog-to-digital converters on both
  - Four Clock management tiles
- Also has two ARM 9 Cores

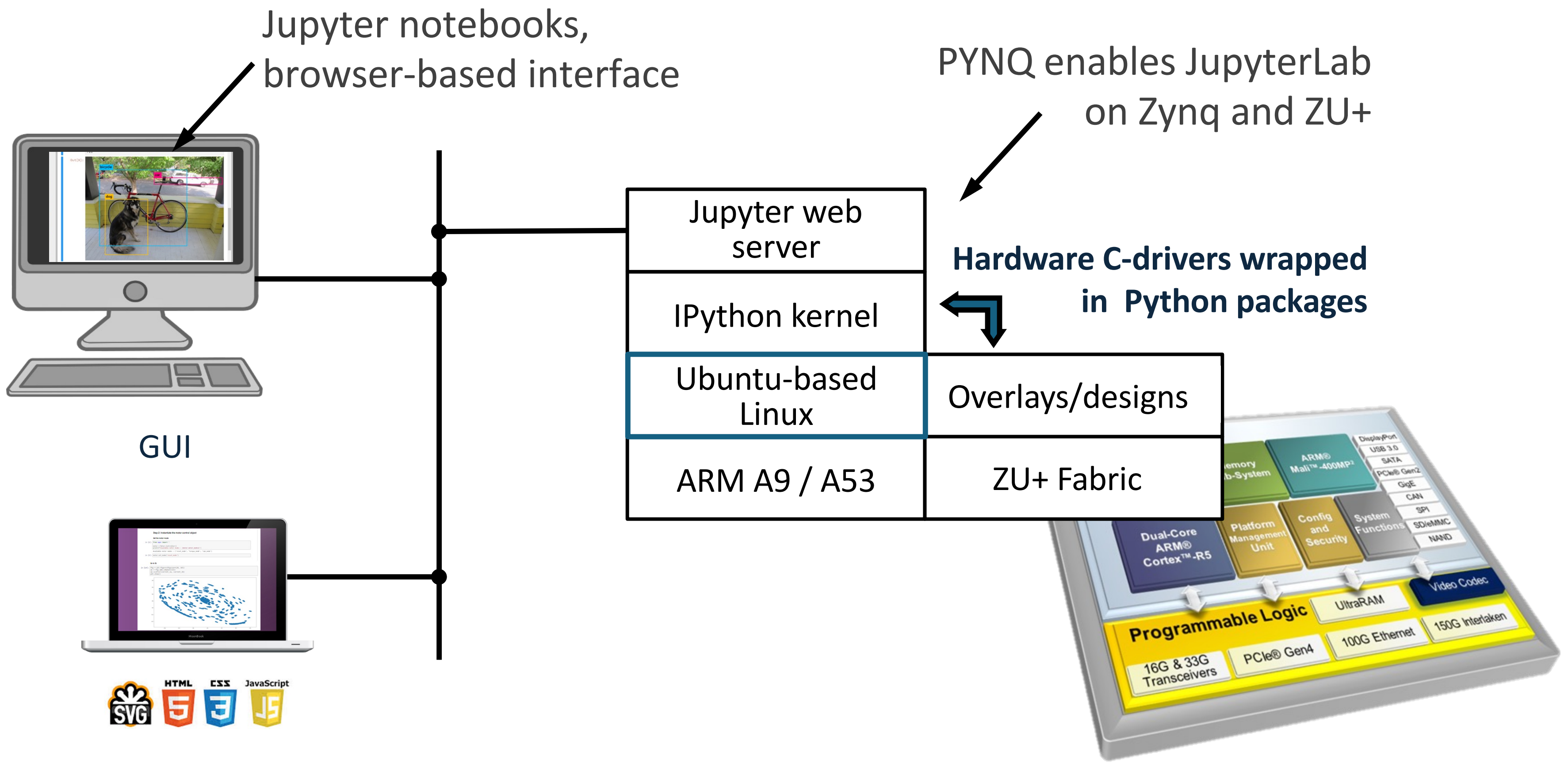
ZYNQ Architecture

- Processing System (PS)
- Programmable Logic (PL)
- Both can be manipulated



Processing System
I/O Unit
Static Memory Controller NOR, NAND, SRAM, QSPI
Dynamic Memory Controller DDR2, DDR3, LPDDR2
2x SPI
2x I2C
2x CAN
2x UART
GPIO
2x SDIO with DMA
2x USB with DMA
2x GigE with DMA
AMBA
ARM® CoreSight™ Multi-core and Trace
NEON™/FPU Engines
NEON™/FPU Engines
Cortex-A9 MPCore™ 32/32 KB I/D Caches
Cortex-A9 MPCore™ 32/32 KB I/D Caches
GIC
Snoop Control Unit (SCU)
ACP
512 KB L2 Cache CTRL
DVC
TTC
WDT
Sys Ctrl Reg
DMA
On-Chip Memory
AXI Interconnect
Programmable Logic:
System Gates, DSP, RAM
Multi Standards IOs (3.3V & High Speed 1.8V)
SysMon/ADC
PCIe
Multi Standards IOs (3.3V & High Speed 1.8V)
Multi Gigabit Transceivers

Python for Zynq…Pynq



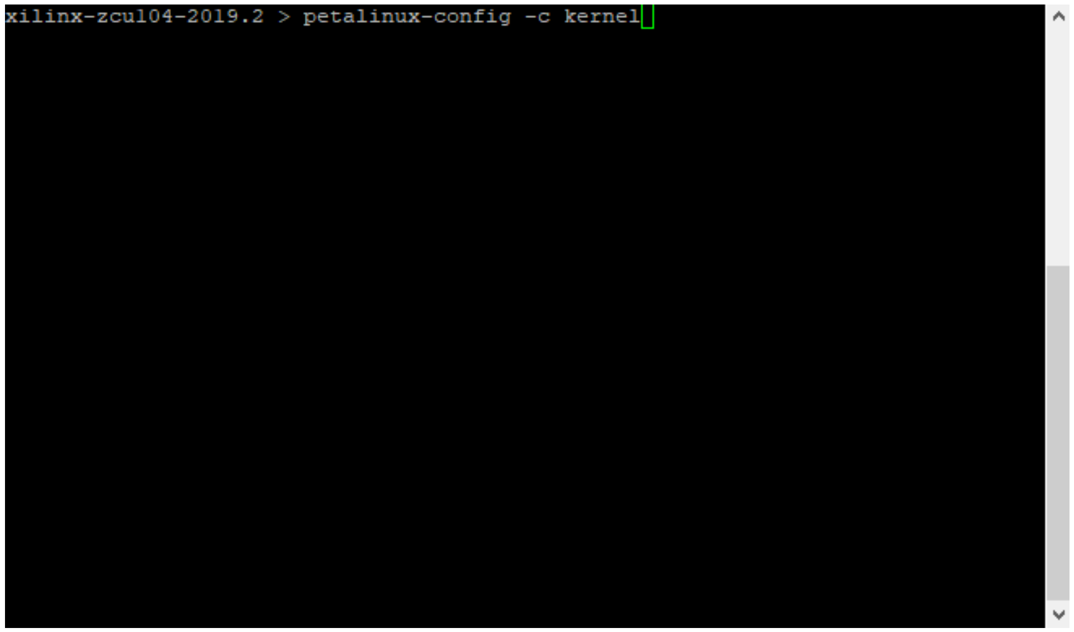
Taken from some Xilinx talk I went to…

Yocto

- Yocto is a project dating back >10 years…focus of it is to build linux for embedded systems applications
- With Yocto you can basically build images of linux distributions targeted at small, particular processors (such as the ARM cores on the Zynq chip)
- Yocto is installed on your computer (kinda like any tool) and then you build for other systems…just like how we build for our FPGA with Vivado.

PetaLinux

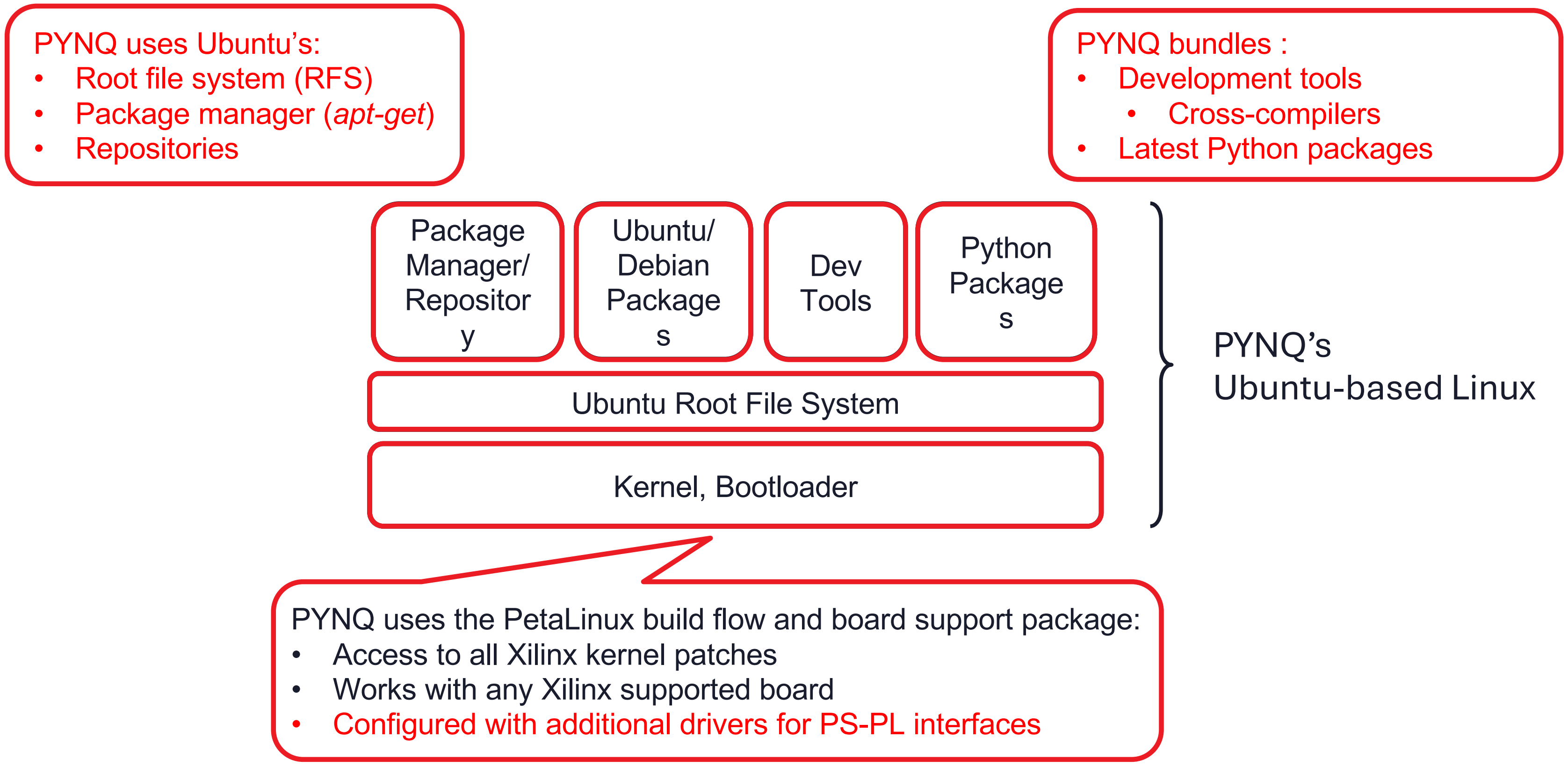
- AMD/Xilinx took Yocto, added some stuff on top intended to streamline these tools for their chips and architectures specifically and called it PetaLinux



```
xilinx-zcu104-2019.2 > petalinux-config -c kernel
```

https://discuss.pynq.io/t/deploying-pynq-and-jupyter-with-petalinux/677

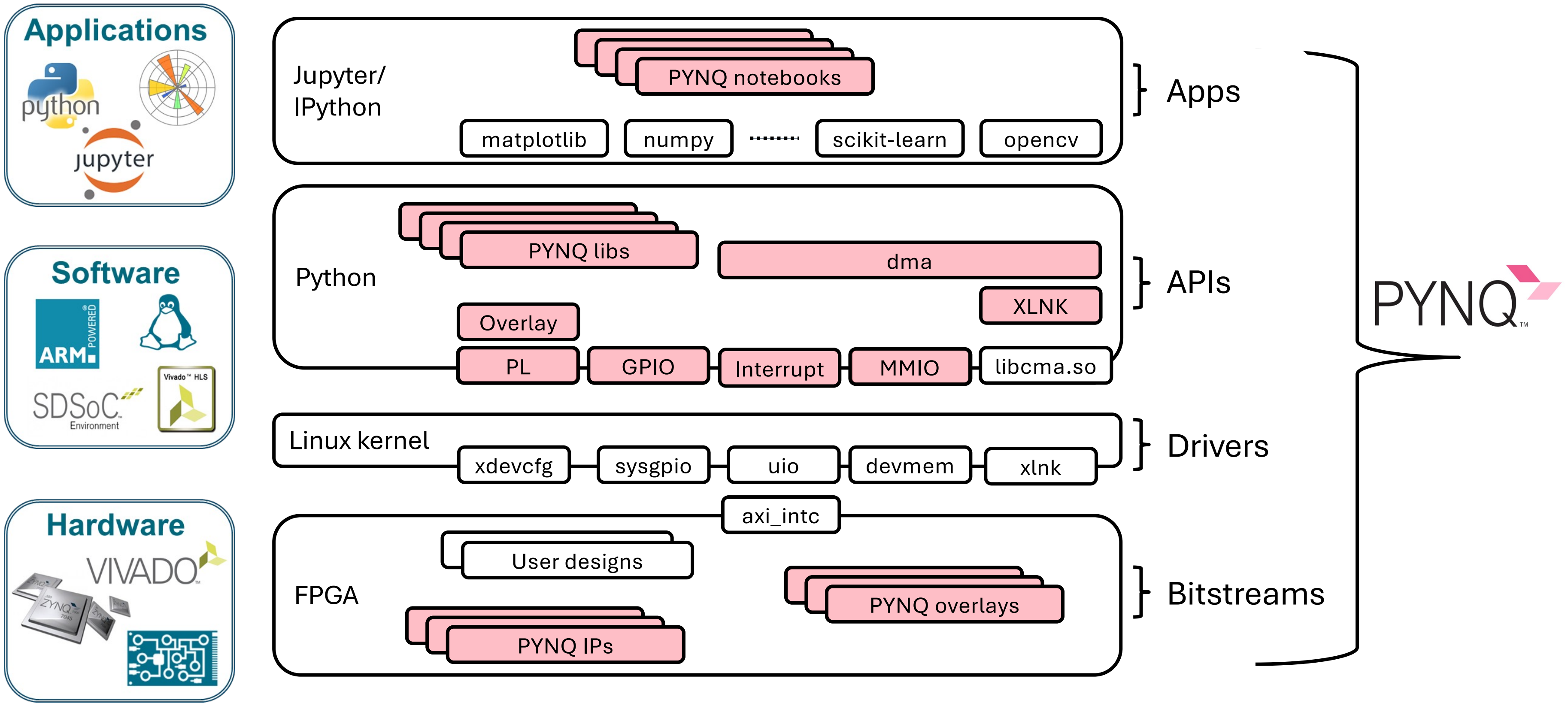
PYNQ uses an Ubuntu based Linux



PYNQ uses Ubuntu's:
Root file system (RFS)
Package manager (*apt-get*)
Repositories
PYNQ bundles :
Development tools
Cross-compilers
Latest Python packages
Package Manager/ Repository
Ubuntu/ Debian Packages
Dev Tools
Python Packages
Ubuntu Root File System
Kernel, Bootloader
PYNQ's Ubuntu-based Linux
PYNQ uses the PetaLinux build flow and board support package:
Access to all Xilinx kernel patches
Works with any Xilinx supported board
Configured with additional drivers for PS-PL interfaces

Taken from some Xilinx talk I went to…

PYNQ Framework

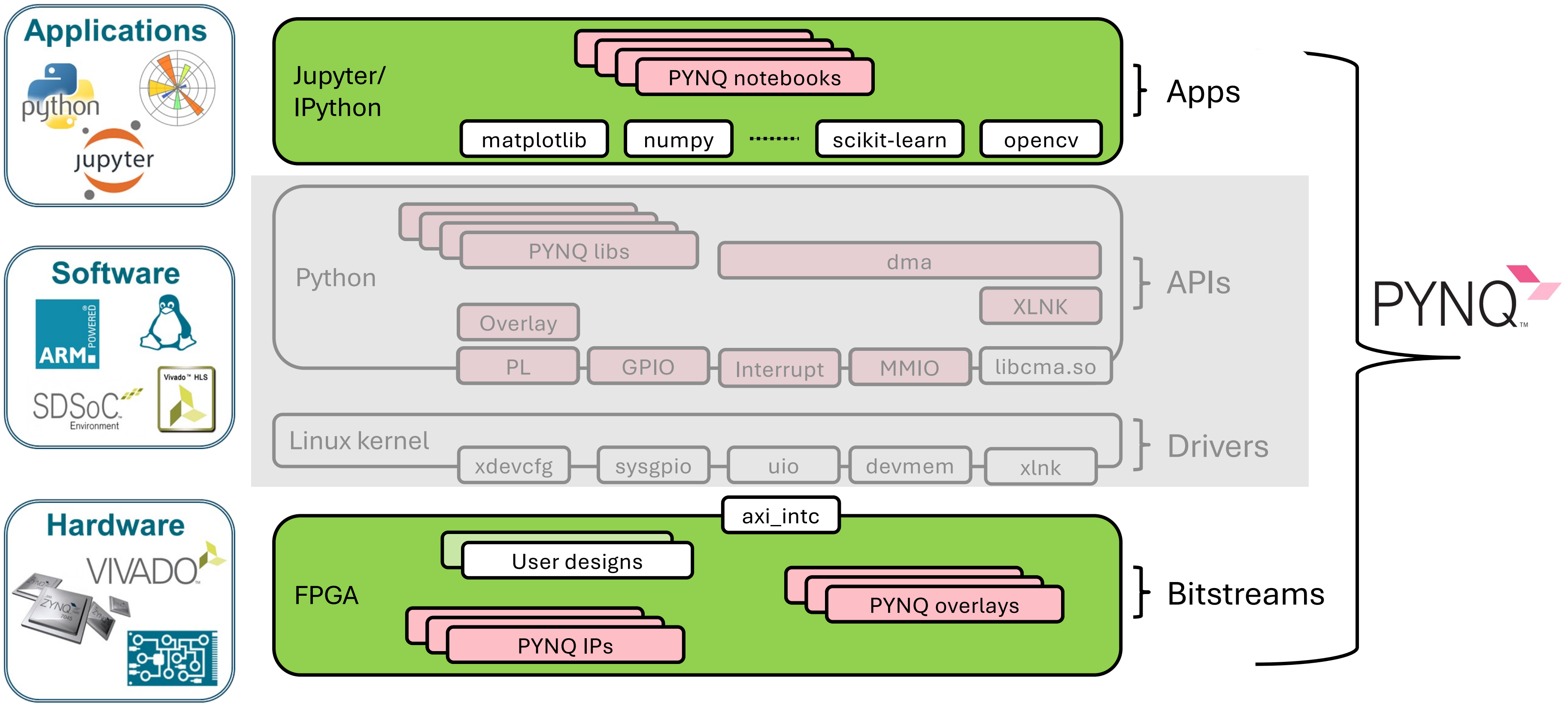


Taken from some Xilinx talk I went to…

PYNQ Compromises

- With the PYNQ framework you're basically starting with a pre-built Yocto/Petalinux implementation that people have already designed for you.
- To get the most out of a chip, one may want to go and do their own custom version and build and then make an image.
- You can 100% build your own pynq image from scratch or with modifications:
  - https://pynq.readthedocs.io/en/latest/pynq_sd_card.html

We’re largely ignoring middle part

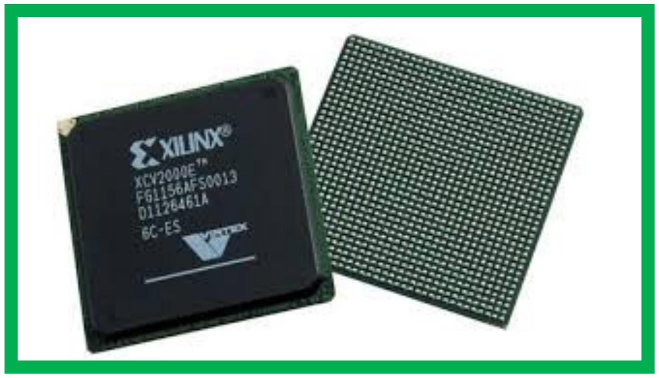


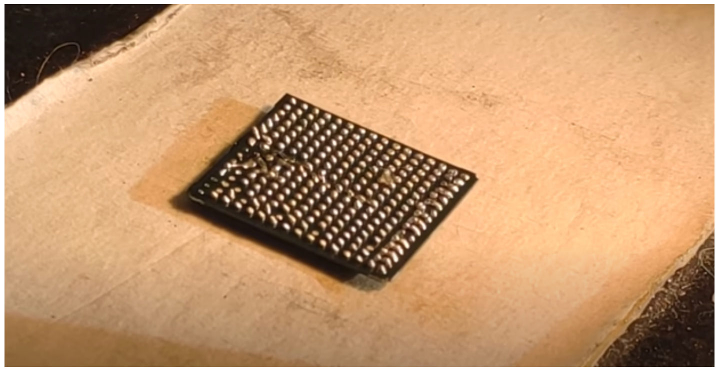
Taken from some Xilinx talk I went to...

ZYNQ 7020 is a chip like any other chip

- Zynq package is a ball grid array (all pins are underneath)
- One of the most unforgiving packages out there…

Can't use iron



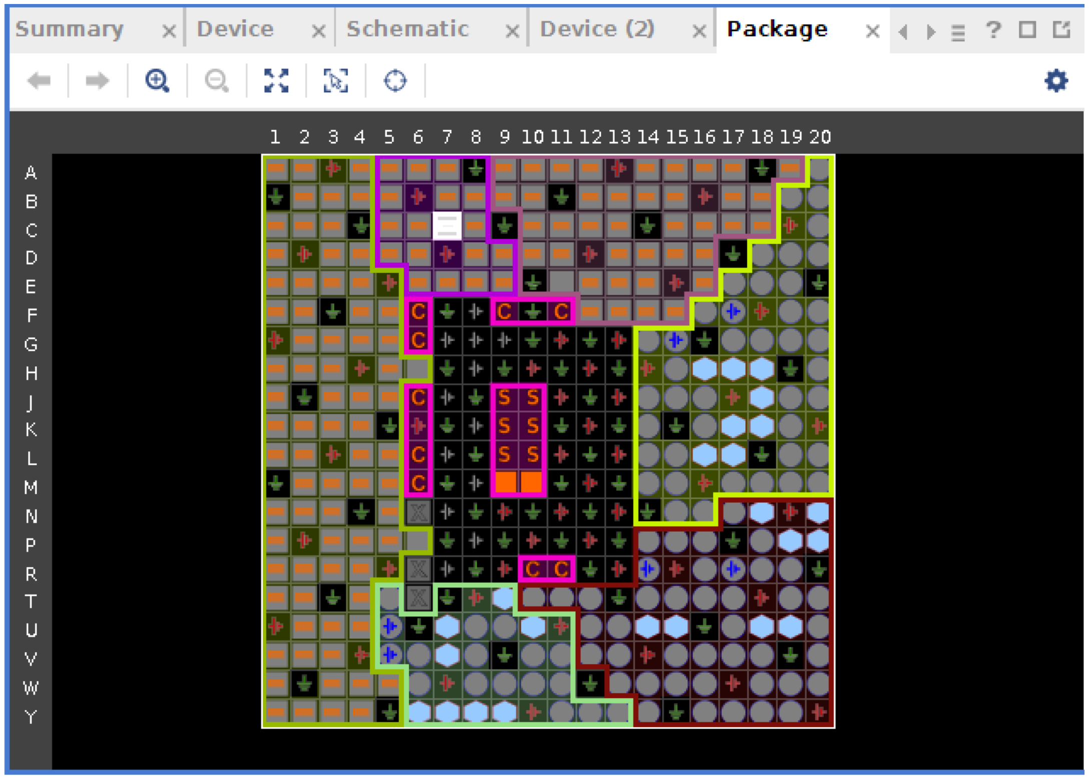


Still from video of somebody "reballing" an Xilinx chip

https://www.youtube.com/watch?v=DVTxHx0z-wo

Assigning Pins

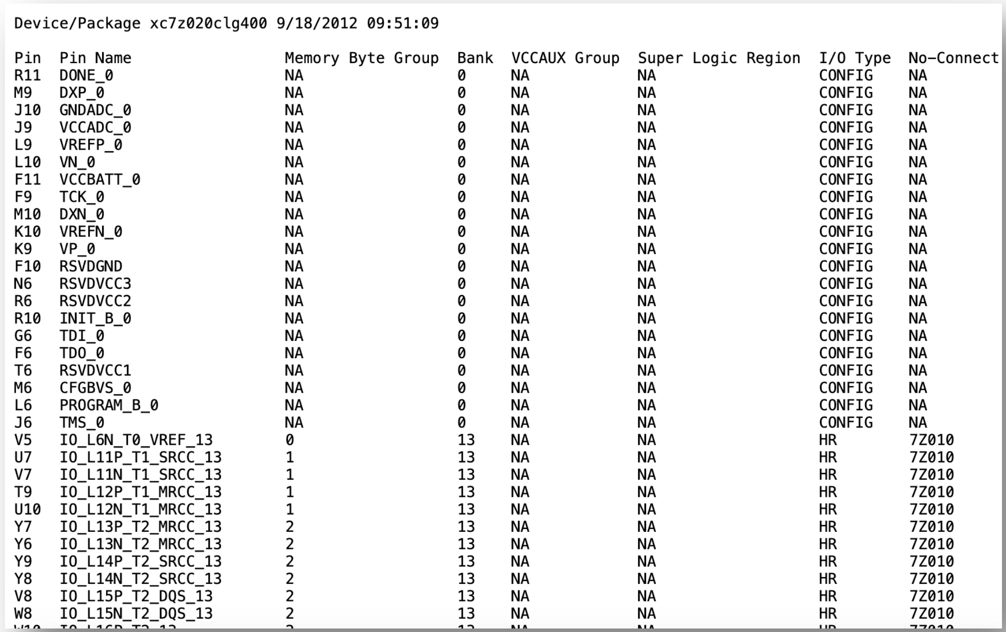
Once design is synthesized you can specify where to route (we'll not do this much since much of this has been decided ahead of time with the PYNQ board's PCB layout, but if you were designing with the chip from scratch this would be part of process



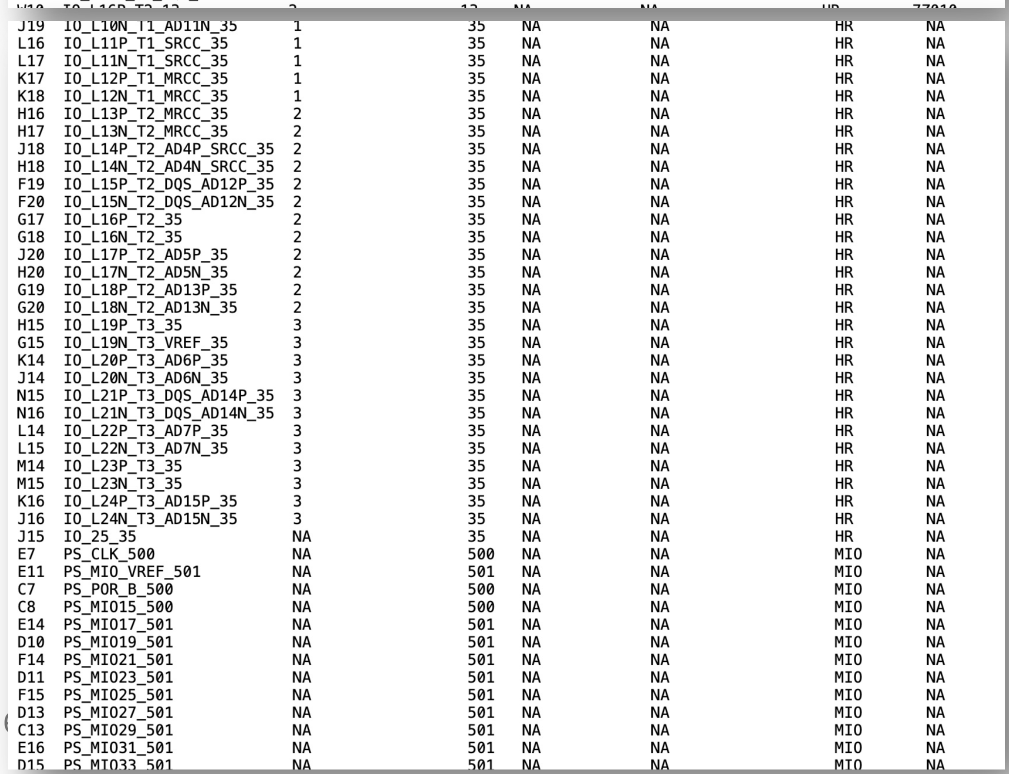
- Pinout file can be found here:
- https://www.xilinx.com/content/dam/xilinx/support/packagefiles/z7packages/xc7z020clg400pkg.txt

400 Pins Listed Out

- Some pins connect to the PL part of chip
- Some pins connect to the PS part of chip.
- Just how it goes…



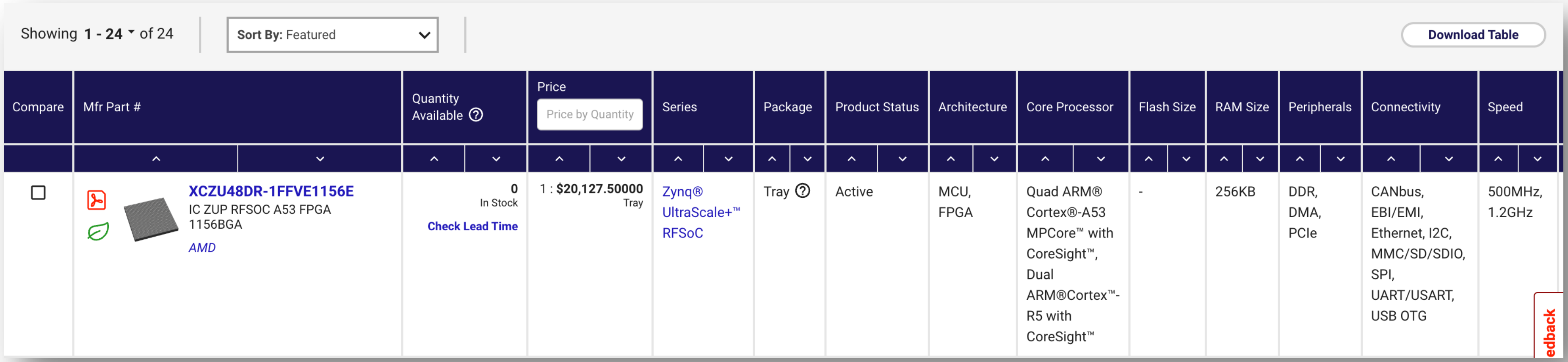
```
Device/Package xc7z020clg400 9/18/2012 09:51:09

Pin  Pin Name                 Memory Byte Group  Bank  VCCAUX Group  Super Logic Region  I/O Type  No-Connect
R11  DONE_0                   NA                 0     NA            NA                  CONFIG    NA
M9   DXP_0                    NA                 0     NA            NA                  CONFIG    NA
J10  GNDADC_0                 NA                 0     NA            NA                  CONFIG    NA
J9   VCCADC_0                 NA                 0     NA            NA                  CONFIG    NA
L9   VREFP_0                  NA                 0     NA            NA                  CONFIG    NA
L10  VN_0                     NA                 0     NA            NA                  CONFIG    NA
F11  VCCBATT_0                NA                 0     NA            NA                  CONFIG    NA
F9   TCK_0                    NA                 0     NA            NA                  CONFIG    NA
M10  DXN_0                    NA                 0     NA            NA                  CONFIG    NA
K10  VREFN_0                  NA                 0     NA            NA                  CONFIG    NA
K9   VP_0                     NA                 0     NA            NA                  CONFIG    NA
F10  RSVDGND                  NA                 0     NA            NA                  CONFIG    NA
N6   RSVDVCC3                 NA                 0     NA            NA                  CONFIG    NA
R6   RSVDVCC2                 NA                 0     NA            NA                  CONFIG    NA
R10  INIT_B_0                 NA                 0     NA            NA                  CONFIG    NA
G6   TDI_0                    NA                 0     NA            NA                  CONFIG    NA
F6   TDO_0                    NA                 0     NA            NA                  CONFIG    NA
T6   RSVDVCC1                 NA                 0     NA            NA                  CONFIG    NA
M6   CFGBVS_0                 NA                 0     NA            NA                  CONFIG    NA
L6   PROGRAM_B_0              NA                 0     NA            NA                  CONFIG    NA
J6   TMS_0                    NA                 0     NA            NA                  CONFIG    NA
V5   IO_L6N_T0_VREF_13        0                  13    NA            NA                  HR        7Z010
U7   IO_L11P_T1_SRCC_13       1                  13    NA            NA                  HR        7Z010
V7   IO_L11N_T1_SRCC_13       1                  13    NA            NA                  HR        7Z010
T9   IO_L12P_T1_MRCC_13       1                  13    NA            NA                  HR        7Z010
U10  IO_L12N_T1_MRCC_13       1                  13    NA            NA                  HR        7Z010
Y7   IO_L13P_T2_MRCC_13       2                  13    NA            NA                  HR        7Z010
Y6   IO_L13N_T2_MRCC_13       2                  13    NA            NA                  HR        7Z010
Y9   IO_L14P_T2_SRCC_13       2                  13    NA            NA                  HR        7Z010
Y8   IO_L14N_T2_SRCC_13       2                  13    NA            NA                  HR        7Z010
V8   IO_L15P_T2_DQS_13        2                  13    NA            NA                  HR        7Z010
W8   IO_L15N_T2_DQS_13        2                  13    NA            NA                  HR        7Z010
```



```
J19  IO_L10N_T1_AD11N_35      1                  35    NA            NA                  HR        NA
L16  IO_L11P_T1_SRCC_35       1                  35    NA            NA                  HR        NA
L17  IO_L11N_T1_SRCC_35       1                  35    NA            NA                  HR        NA
K17  IO_L12P_T1_MRCC_35       1                  35    NA            NA                  HR        NA
K18  IO_L12N_T1_MRCC_35       1                  35    NA            NA                  HR        NA
H16  IO_L13P_T2_MRCC_35       2                  35    NA            NA                  HR        NA
H17  IO_L13N_T2_MRCC_35       2                  35    NA            NA                  HR        NA
J18  IO_L14P_T2_AD4P_SRCC_35  2                  35    NA            NA                  HR        NA
H18  IO_L14N_T2_AD4N_SRCC_35  2                  35    NA            NA                  HR        NA
F19  IO_L15P_T2_DQS_AD12P_35  2                  35    NA            NA                  HR        NA
F20  IO_L15N_T2_DQS_AD12N_35  2                  35    NA            NA                  HR        NA
G17  IO_L16P_T2_35            2                  35    NA            NA                  HR        NA
G18  IO_L16N_T2_35            2                  35    NA            NA                  HR        NA
J20  IO_L17P_T2_AD5P_35       2                  35    NA            NA                  HR        NA
H20  IO_L17N_T2_AD5N_35       2                  35    NA            NA                  HR        NA
G19  IO_L18P_T2_AD13P_35      2                  35    NA            NA                  HR        NA
G20  IO_L18N_T2_AD13N_35      2                  35    NA            NA                  HR        NA
H15  IO_L19P_T3_35            3                  35    NA            NA                  HR        NA
G15  IO_L19N_T3_VREF_35       3                  35    NA            NA                  HR        NA
K14  IO_L20P_T3_AD6P_35       3                  35    NA            NA                  HR        NA
J14  IO_L20N_T3_AD6N_35       3                  35    NA            NA                  HR        NA
N15  IO_L21P_T3_DQS_AD14P_35  3                  35    NA            NA                  HR        NA
N16  IO_L21N_T3_DQS_AD14N_35  3                  35    NA            NA                  HR        NA
L14  IO_L22P_T3_AD7P_35       3                  35    NA            NA                  HR        NA
L15  IO_L22N_T3_AD7N_35       3                  35    NA            NA                  HR        NA
M14  IO_L23P_T3_35            3                  35    NA            NA                  HR        NA
M15  IO_L23N_T3_35            3                  35    NA            NA                  HR        NA
K16  IO_L24P_T3_AD15P_35      3                  35    NA            NA                  HR        NA
J16  IO_L24N_T3_AD15N_35      3                  35    NA            NA                  HR        NA
J15  IO_25_35                 NA                 35    NA            NA                  HR        NA
E7   PS_CLK_500               NA                 500   NA            NA                  MIO       NA
E11  PS_MIO_VREF_501          NA                 501   NA            NA                  MIO       NA
C7   PS_POR_B_500             NA                 500   NA            NA                  MIO       NA
C8   PS_MIO15_500             NA                 500   NA            NA                  MIO       NA
E14  PS_MIO17_501             NA                 501   NA            NA                  MIO       NA
D10  PS_MIO19_501             NA                 501   NA            NA                  MIO       NA
F14  PS_MIO21_501             NA                 501   NA            NA                  MIO       NA
D11  PS_MIO23_501             NA                 501   NA            NA                  MIO       NA
F15  PS_MIO25_501             NA                 501   NA            NA                  MIO       NA
D13  PS_MIO27_501             NA                 501   NA            NA                  MIO       NA
C13  PS_MIO29_501             NA                 501   NA            NA                  MIO       NA
E16  PS_MIO31_501             NA                 501   NA            NA                  MIO       NA
D15  PS_MIO33_501             NA                 501   NA            NA                  MIO       NA
```

Aside…The RFSoC is Bigger

- Go to this site (https://www.xilinx.com/support/package-pinout-files/zynq-ultrascale-plus-pkgs.html) and use the non-functional sort tools to find the pin file for the xczu48
- You’ll see that it is a 1156 pin BGA

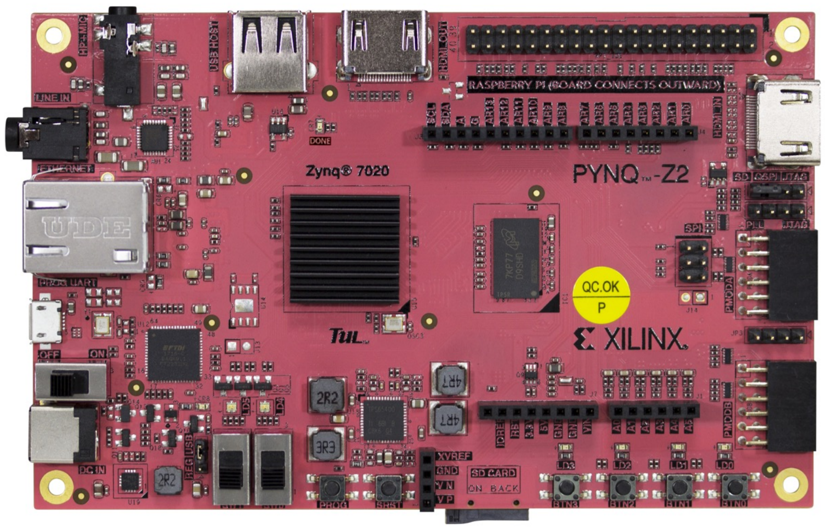


Showing 1 - 24 of 24 | Sort By: Featured | Download Table

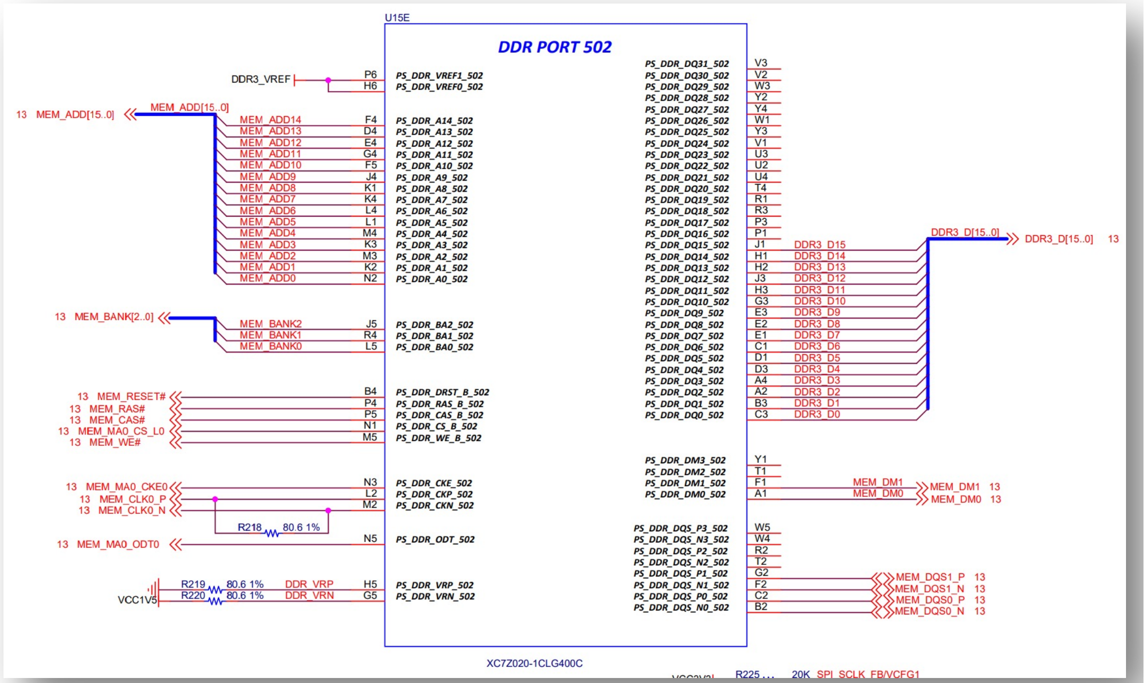
Compare	Mfr Part #	Quantity Available	Price	Series	Package	Product Status	Architecture	Core Processor	Flash Size	RAM Size	Peripherals	Connectivity	Speed
☐	XCZU48DR-1FFVE1156E IC ZUP RFSOC A53 FPGA 1156BGA *AMD*	0 In Stock Check Lead Time	1 : $20,127.50000 Tray	Zynq® UltraScale+™ RFSoC	Tray	Active	MCU, FPGA	Quad ARM® Cortex®-A53 MPCore™ with CoreSight™, Dual ARM®Cortex™-R5 with CoreSight™	-	256KB	DDR, DMA, PCIe	CANbus, EBI/EMI, Ethernet, I2C, MMC/SD/SDIO, SPI, UART/USART, USB OTG	500MHz, 1.2GHz

Now, the Pynq Z2 board made some choices for us

- If you were the engineer laying out the chip/board from scratch you would also need to make these decision.
- Some decisions have very little wiggle room, others do.



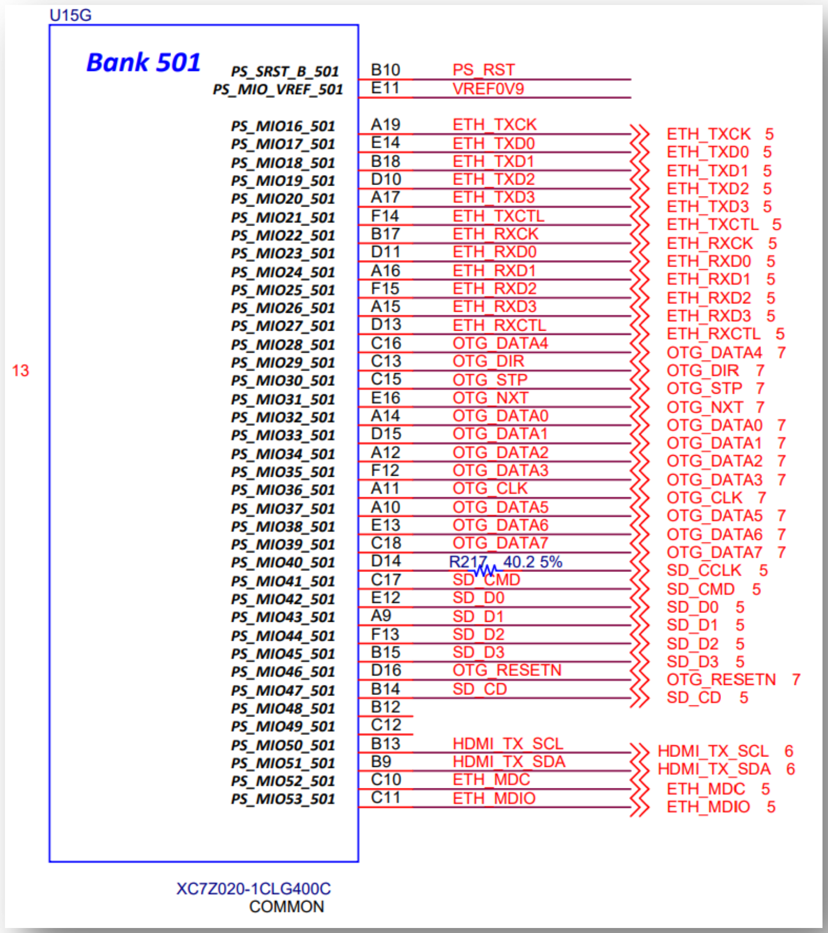
Schematic of PYNQ Z2 *Board*



- The 512 MB DRAM is routed to PS_... Pins of the Zynq chip.
- Meaning the DRAM is only accessible in the processing side

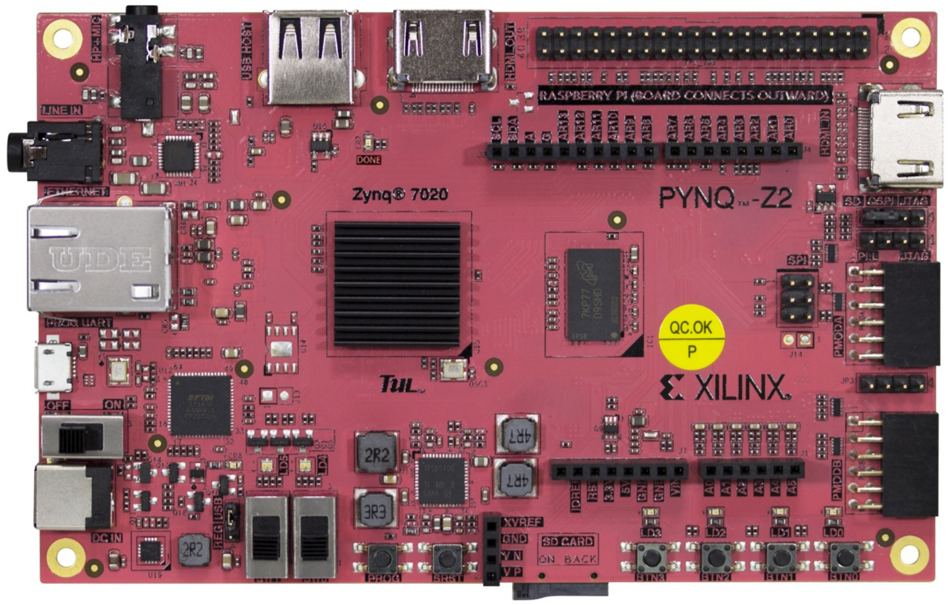
Schematic of PYNQ Z2 *Board*

- Ethernet, SD card, some HDMI control portions, OTG/USB are all also wired to PS_ pins
- That means those are not accessible via


U15G
Bank 501
13
PS_SRST_B_501 B10 PS_RST
PS_MIO_VREF_501 E11 VREF0V9
PS_MIO16_501 A19 ETH_TXCK ETH_TXCK 5
PS_MIO17_501 E14 ETH_TXD0 ETH_TXD0 5
PS_MIO18_501 B18 ETH_TXD1 ETH_TXD1 5
PS_MIO19_501 D10 ETH_TXD2 ETH_TXD2 5
PS_MIO20_501 A17 ETH_TXD3 ETH_TXD3 5
PS_MIO21_501 F14 ETH_TXCTL ETH_TXCTL 5
PS_MIO22_501 B17 ETH_RXCK ETH_RXCK 5
PS_MIO23_501 D11 ETH_RXD0 ETH_RXD0 5
PS_MIO24_501 A16 ETH_RXD1 ETH_RXD1 5
PS_MIO25_501 F15 ETH_RXD2 ETH_RXD2 5
PS_MIO26_501 A15 ETH_RXD3 ETH_RXD3 5
PS_MIO27_501 D13 ETH_RXCTL ETH_RXCTL 5
PS_MIO28_501 C16 OTG_DATA4 OTG_DATA4 7
PS_MIO29_501 C13 OTG_DIR OTG_DIR 7
PS_MIO30_501 C15 OTG_STP OTG_STP 7
PS_MIO31_501 E16 OTG_NXT OTG_NXT 7
PS_MIO32_501 A14 OTG_DATA0 OTG_DATA0 7
PS_MIO33_501 D15 OTG_DATA1 OTG_DATA1 7
PS_MIO34_501 A12 OTG_DATA2 OTG_DATA2 7
PS_MIO35_501 F12 OTG_DATA3 OTG_DATA3 7
PS_MIO36_501 A11 OTG_CLK OTG_CLK 7
PS_MIO37_501 A10 OTG_DATA5 OTG_DATA5 7
PS_MIO38_501 E13 OTG_DATA6 OTG_DATA6 7
PS_MIO39_501 C18 OTG_DATA7 OTG_DATA7 7
PS_MIO40_501 D14 R217 40.2 5% SD_CCLK 5
PS_MIO41_501 C17 SD_CMD SD_CMD 5
PS_MIO42_501 E12 SD_D0 SD_D0 5
PS_MIO43_501 A9 SD_D1 SD_D1 5
PS_MIO44_501 F13 SD_D2 SD_D2 5
PS_MIO45_501 B15 SD_D3 SD_D3 5
PS_MIO46_501 D16 OTG_RESETN OTG_RESETN 7
PS_MIO47_501 B14 SD_CD SD_CD 5
PS_MIO48_501 B12
PS_MIO49_501 C12
PS_MIO50_501 B13 HDMI_TX_SCL HDMI_TX_SCL 6
PS_MIO51_501 B9 HDMI_TX_SDA HDMI_TX_SDA 6
PS_MIO52_501 C10 ETH_MDC ETH_MDC 5
PS_MIO53_501 C11 ETH_MDIO ETH_MDIO 5
XC7Z020-1CLG400C
COMMON

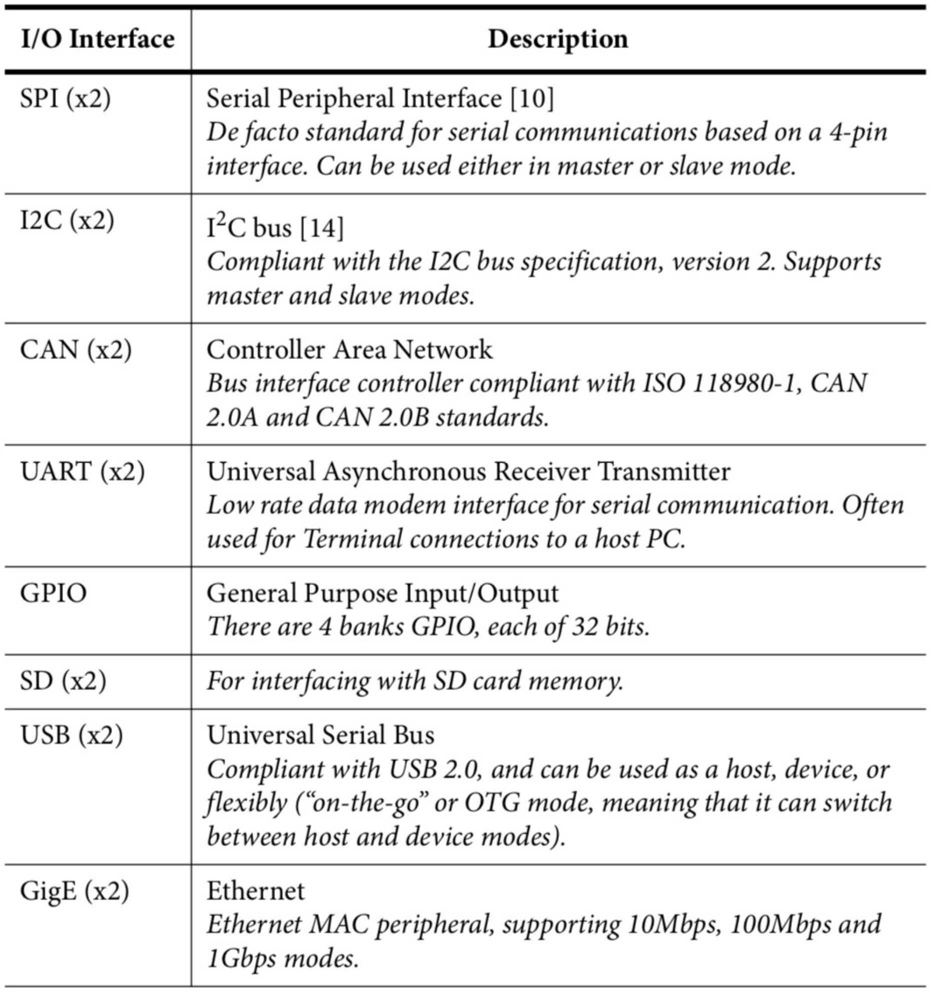
Most other things on the board are actually wired to pins that are part of the PL (Programmable Logic)

- So pretty much everything else…
- All these the random pins, the audio, the HDMI in/out, buttons, etc…



List of I/O Peripherals for the PS:

- ”Hard” IP cores exist on the **PS** that perform certain interfacing roles/protocols:
- These can be multiplexed out to many subsets of pins



I/O Interface	Description
SPI (x2)	Serial Peripheral Interface [10] *De facto standard for serial communications based on a 4-pin interface. Can be used either in master or slave mode.*
I2C (x2)	I^2C bus [14] *Compliant with the I2C bus specification, version 2. Supports master and slave modes.*
CAN (x2)	Controller Area Network *Bus interface controller compliant with ISO 118980-1, CAN 2.0A and CAN 2.0B standards.*
UART (x2)	Universal Asynchronous Receiver Transmitter *Low rate data modem interface for serial communication. Often used for Terminal connections to a host PC.*
GPIO	General Purpose Input/Output *There are 4 banks GPIO, each of 32 bits.*
SD (x2)	*For interfacing with SD card memory.*
USB (x2)	Universal Serial Bus *Compliant with USB 2.0, and can be used as a host, device, or flexibly (“on-the-go” or OTG mode, meaning that it can switch between host and device modes).*
GigE (x2)	Ethernet *Ethernet MAC peripheral, supporting 10Mbps, 100Mbps and 1Gbps modes.*

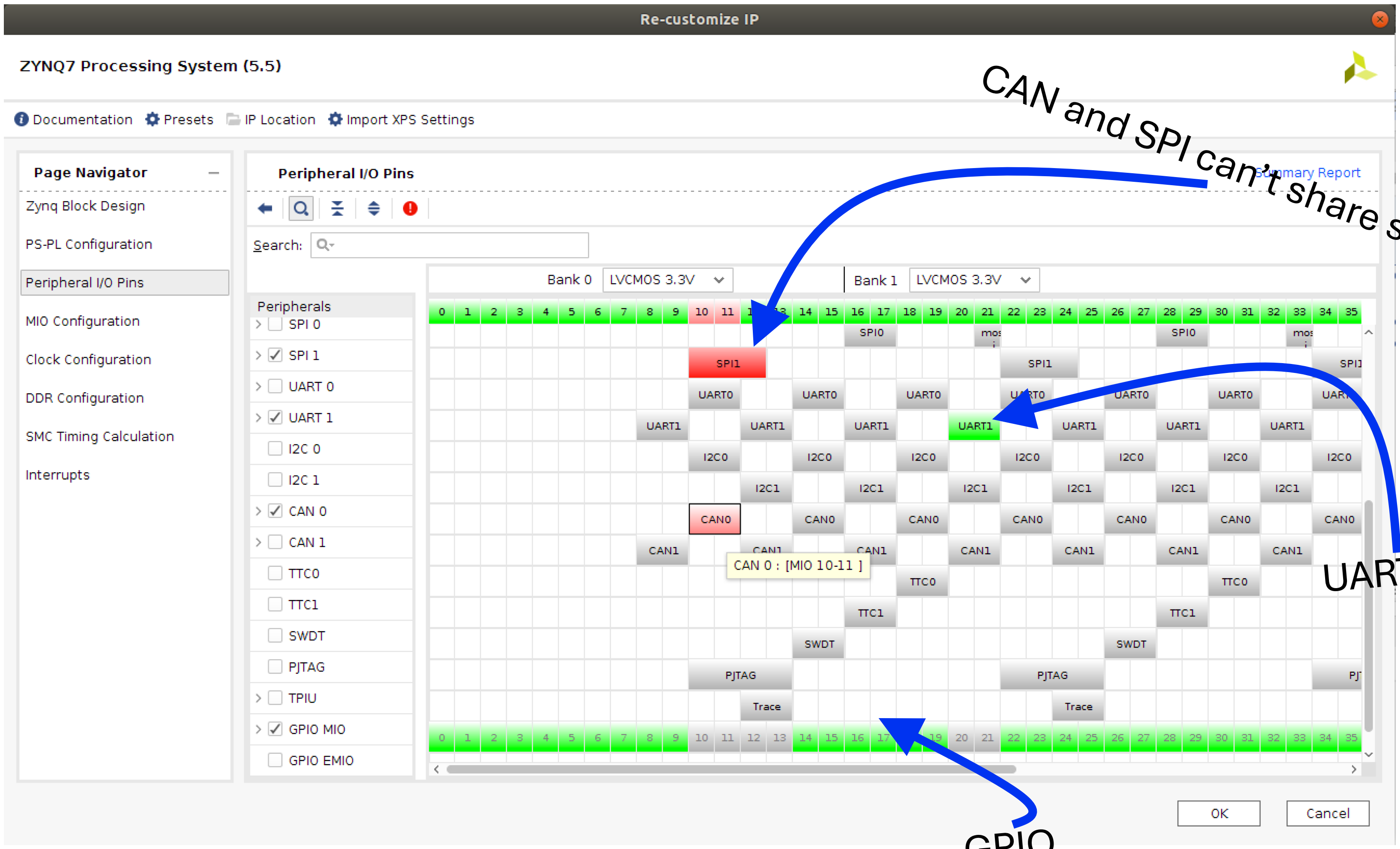
Taken from The Zynq Book

Using them

- In a normal microcontroller, you would simply activate a module, such as an SPI controller and connect it to some pins.
- The way the Pynq Z2 board is laid out you can’t do that.
- In an effort to ensure flexibility for development, they connected most things and broke out most general IO from the PL side.

Assigning I/O pins to Hard IP Peripherals

Here I double-clicked on the Zynq7 Processing IP Core



Re-customize IP

ZYNQ7 Processing System (5.5)

Documentation · Presets · IP Location · Import XPS Settings

Page Navigator

- Zynq Block Design
- PS-PL Configuration
- Peripheral I/O Pins
- MIO Configuration
- Clock Configuration
- DDR Configuration
- SMC Timing Calculation
- Interrupts

Peripheral I/O Pins

Summary Report

Search:

Bank 0: LVCMOS 3.3V · Bank 1: LVCMOS 3.3V

Peripherals
☐ SPI 0
☑ SPI 1
☐ UART 0
☑ UART 1
☐ I2C 0
☐ I2C 1
☑ CAN 0
☐ CAN 1
☐ TTC0
☐ TTC1
☐ SWDT
☐ PJTAG
☐ TPIU
☑ GPIO MIO
☐ GPIO EMIO

CAN 0 : [MIO 10-11]

OK · Cancel

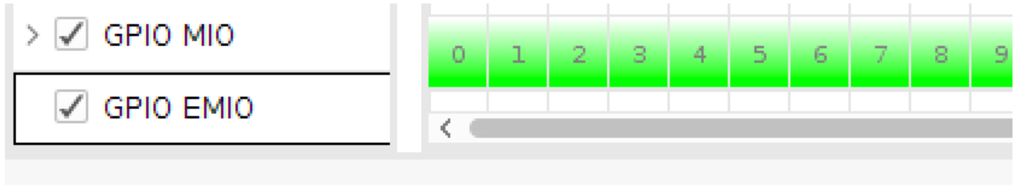
CAN and SPI can't share same pins!

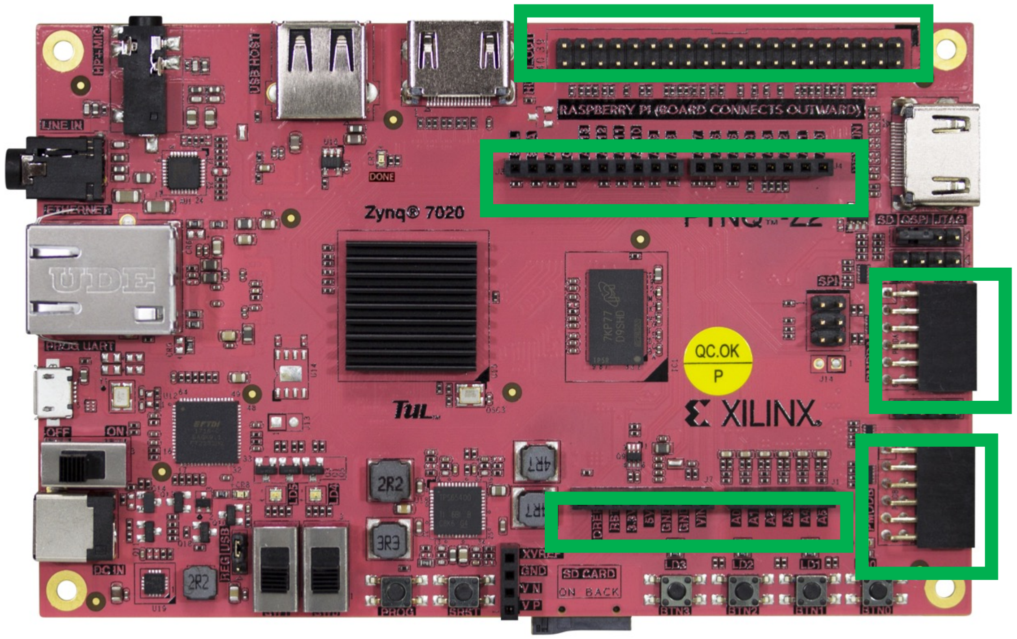
UART is fine

GPIO

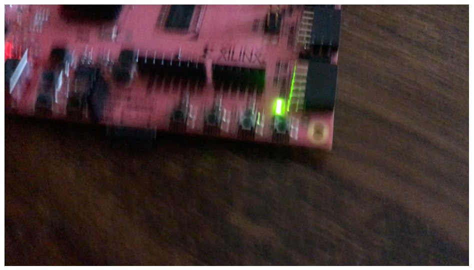
Linking to Outside World

- The I/O pins normally go to the outside world, but on our PYNQ board we need to extend them into the PL (which has its own actual physical output pins)
- Making the GPIO pins **EMIO** (Extended) Multiplexed In/Out) puts them into the PL for further manipulation



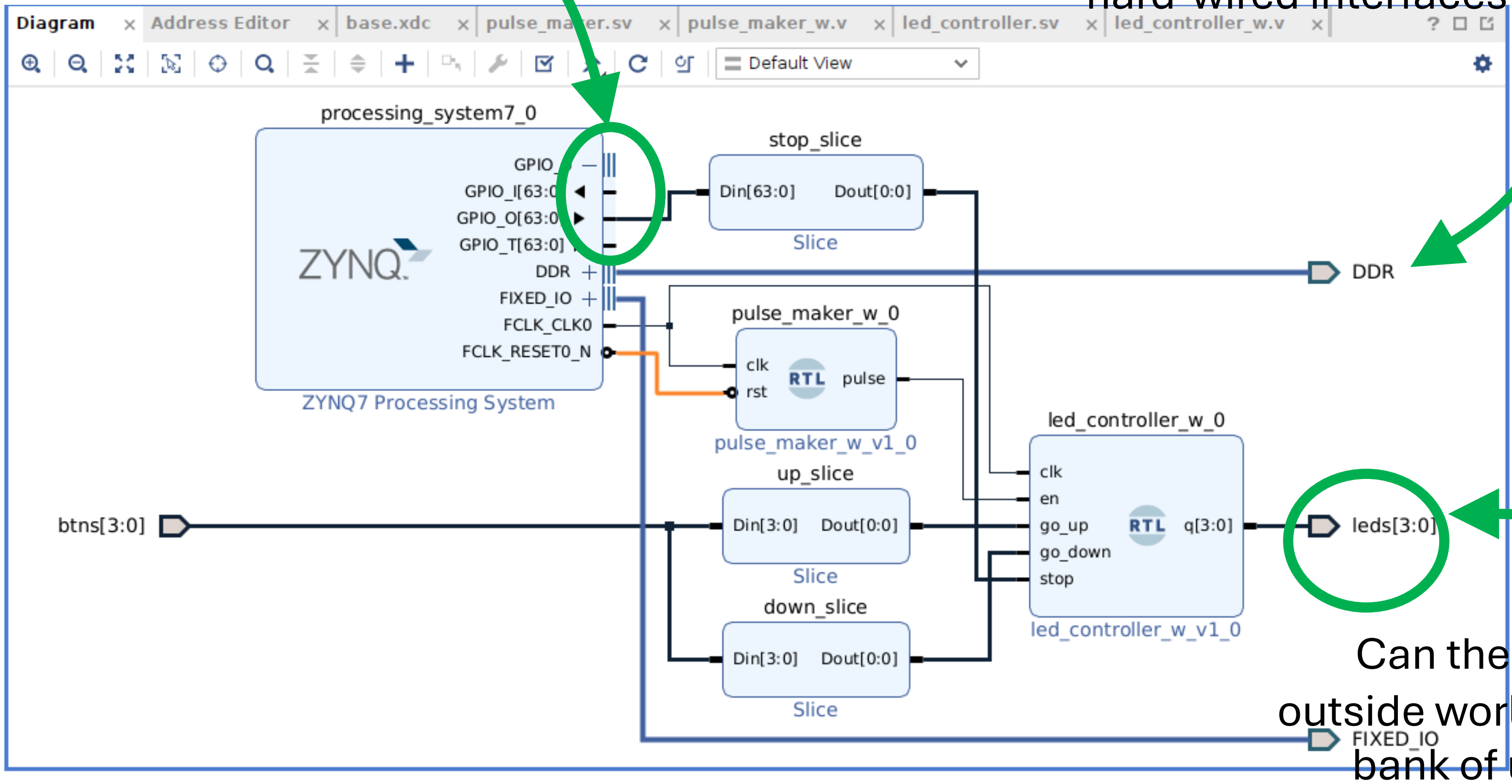


Lab 1



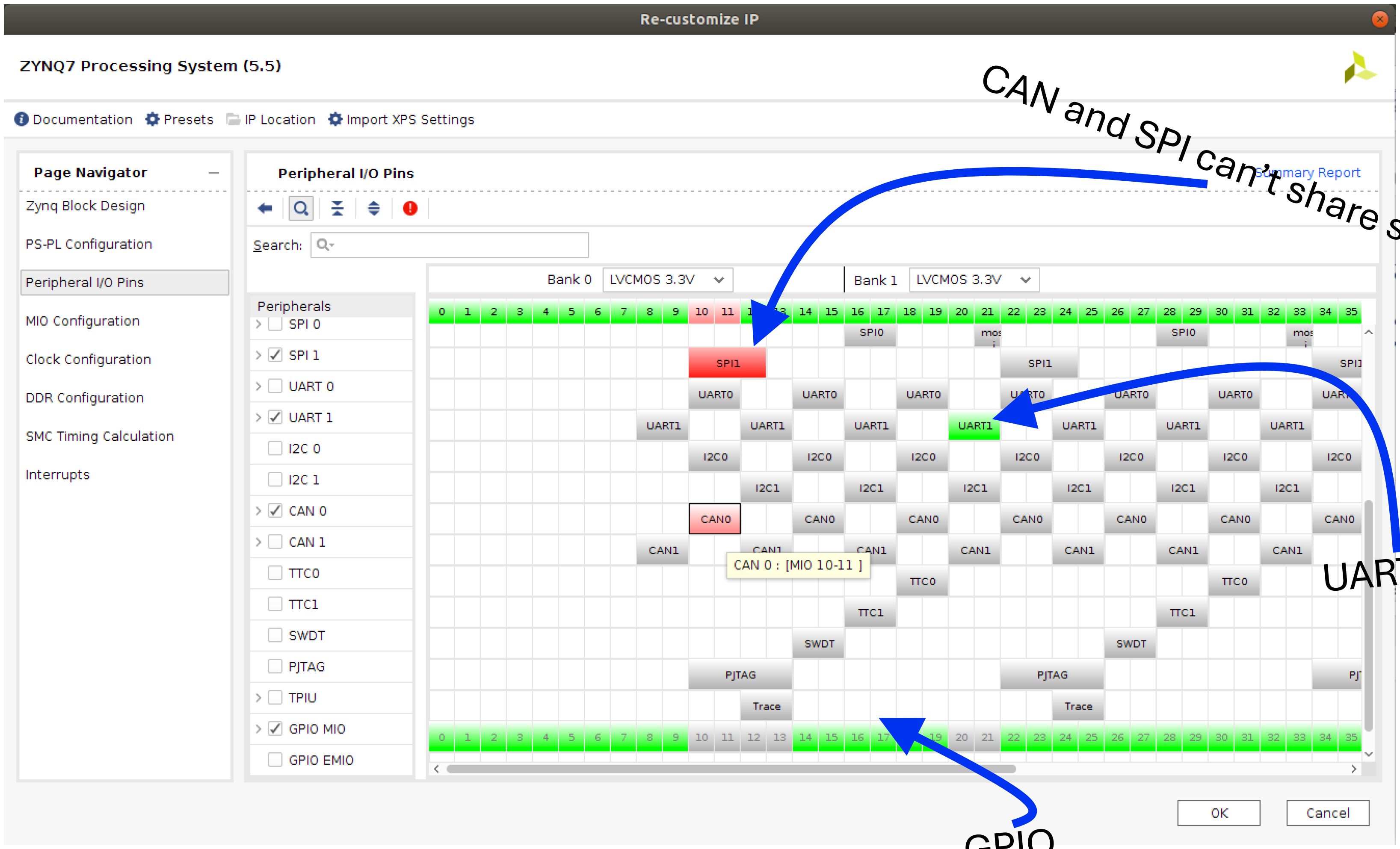
We have specified the Zynq PS to route its IO pins out into the PL fabric and we can do what we want with them

These represent pins that come directly from the PS and interface with DRAM (DDR) and some hard-wired interfaces



Can then route into outside world from PL's bank of usable pins

Clicking on these things is really just a nice way to configure internal multiplexers



Bank 0 LVCMOS 3.3V

Bank 1 LVCMOS 3.3V

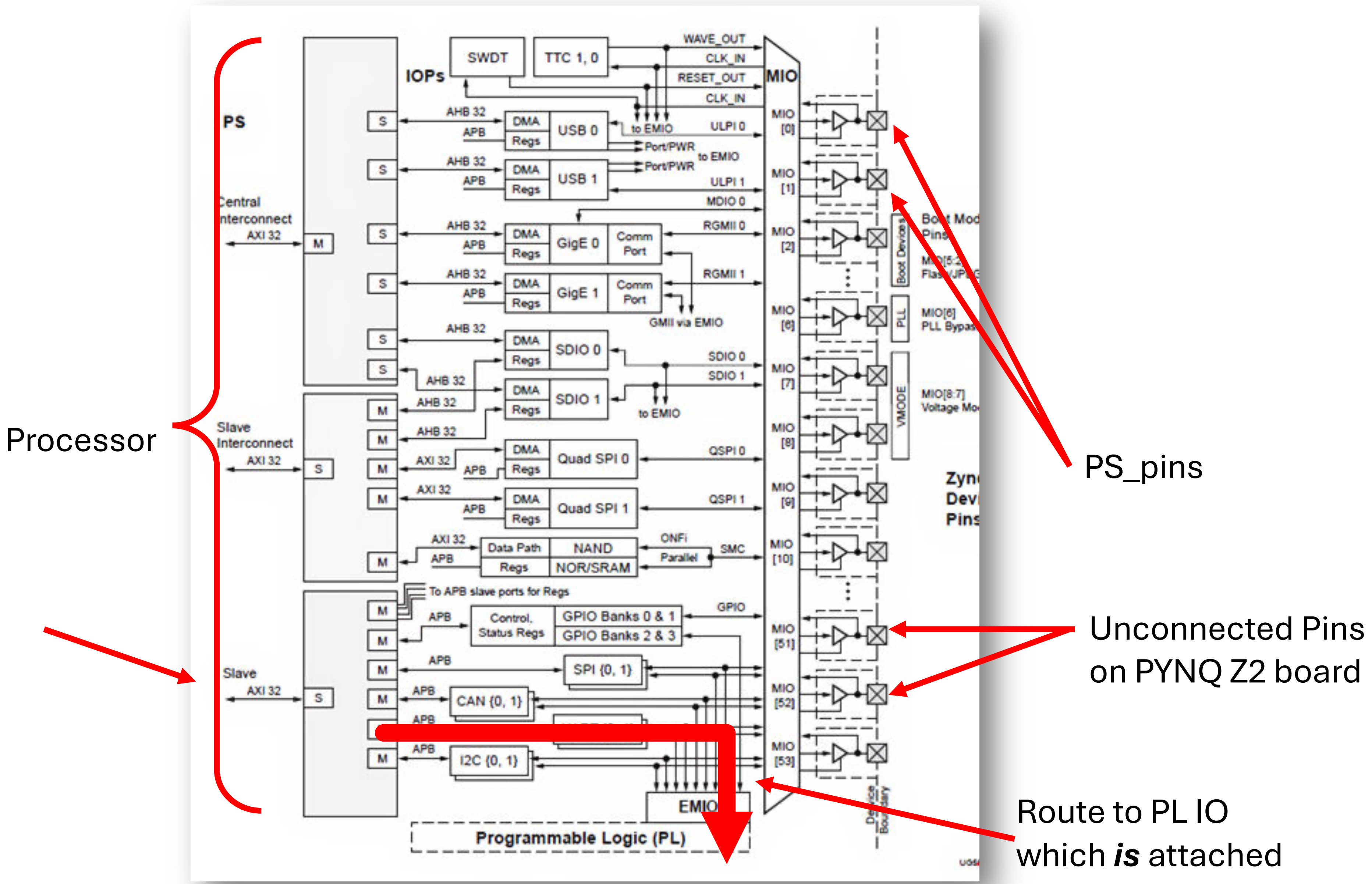
Peripheral	Enabled
SPI 0	☐
SPI 1	✓
UART 0	☐
UART 1	✓
I2C 0	☐
I2C 1	☐
CAN 0	✓
CAN 1	☐
TTC0	☐
TTC1	☐
SWDT	☐
PJTAG	☐
TPIU	☐
GPIO MIO	✓
GPIO EMIO	☐

CAN 0 : [MIO 10-11]

CAN and SPI can't share same pins!

UART is fine

GPIO



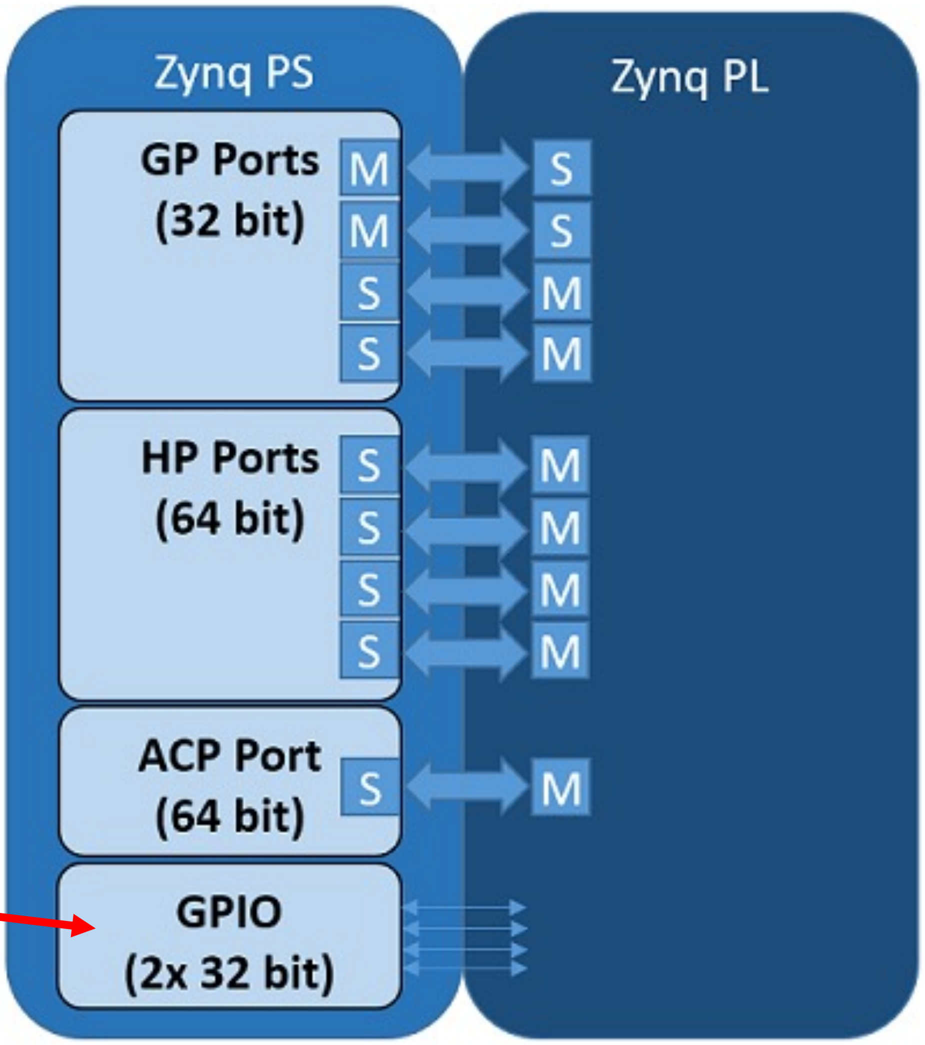
Taken from the MicroZed Chronicles Blog/Xilinx Docs

Other PL-PS Interconnects

Interface Between PS and PL

- Four Ways to Transfer Data from the PS to the PL
  - 64 bits of GPIO
  - 4 GP AXI Ports
  - 4 HP AXI Ports
  - 1 ACP Port

Just talked about this

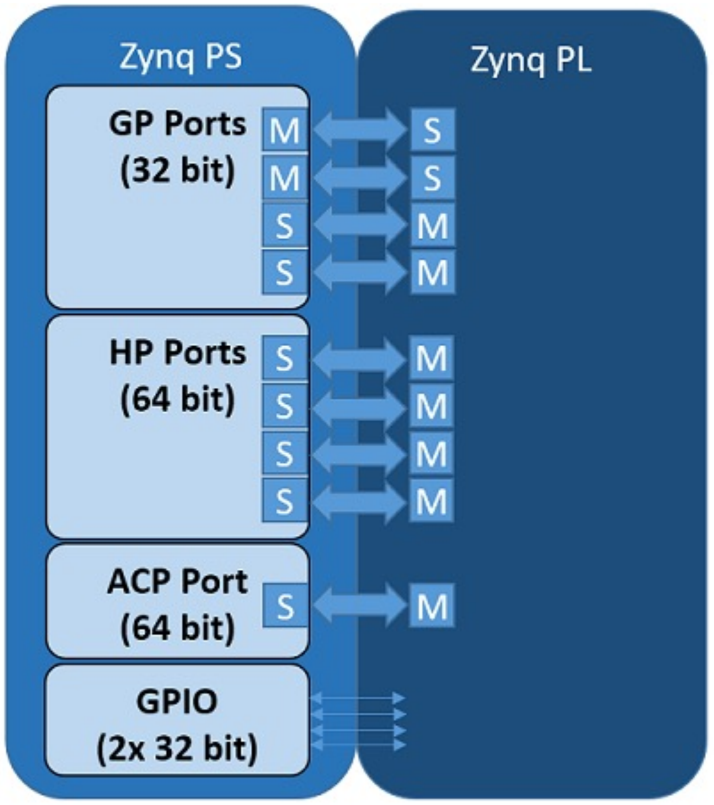


https://pynq.readthedocs.io/en/v2.3/overlay_design_methodology/pspl_interface.html

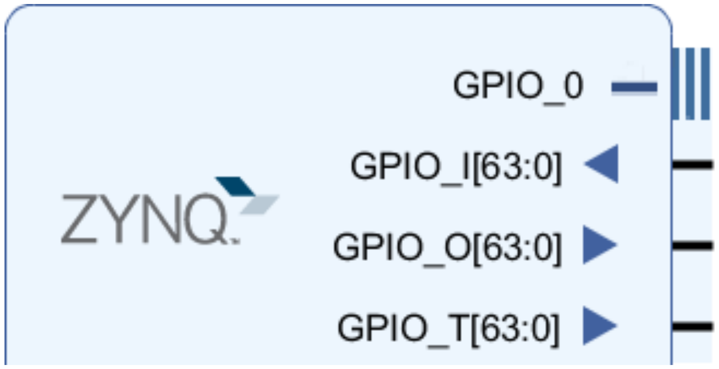
GPIO Pins

- **G**eneral **P**urpose **I**nput **O**utput

- You can via software (writing to registers), control and be controlled by ~54 pins
- These are good for low-speed control, configuration, reset signals…things like that.



Interrupts

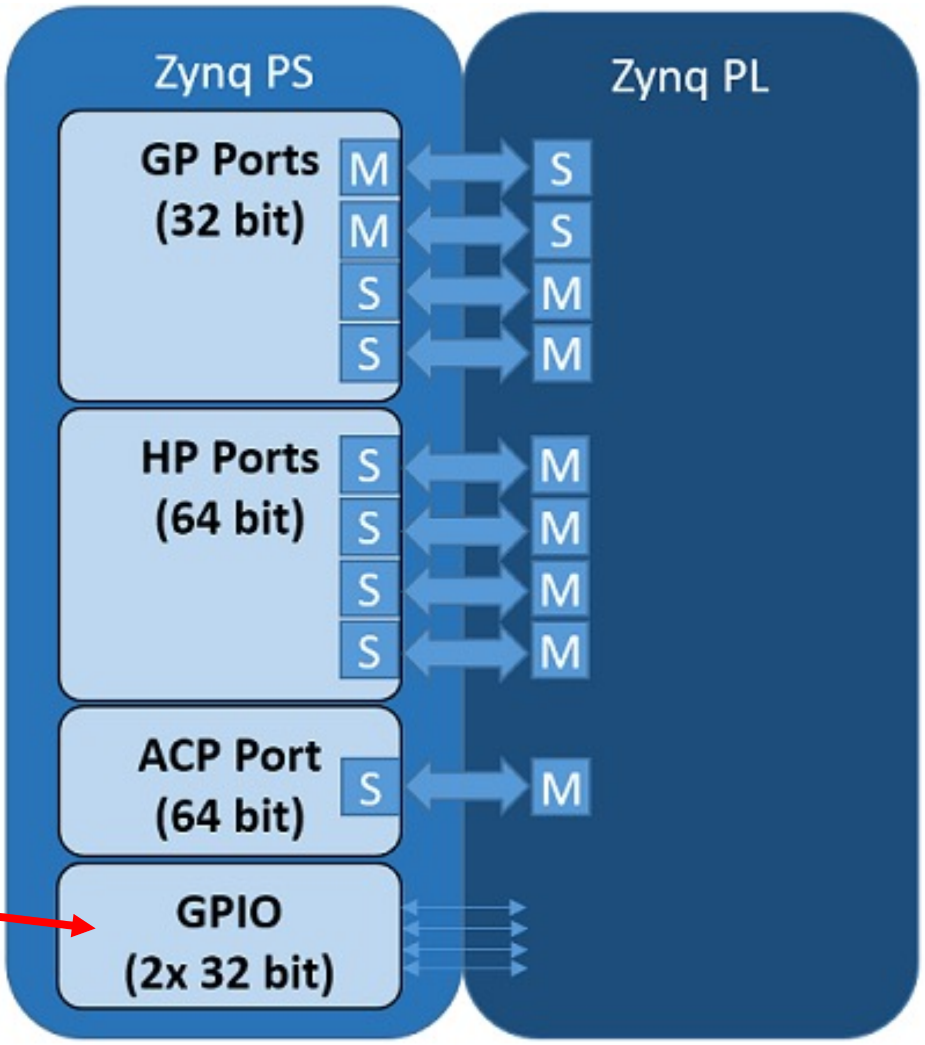


- The GPIO of the PS can be setup to have interrupts even when you are routing them "internally" into the PL Using EMIO.
- This means you can actually have the PL trigger Python processes to run by setting up the interrupts as well as some async programming on the Python side
- https://pynq.readthedocs.io/en/latest/pynq_libraries/interrupt.html
- https://pynq.readthedocs.io/en/latest/overlay_design_methodology/pynq_and_asyncio.html#pynq-and-asyncio

Interface Between PS and PL

- Four Ways to Transfer Data from the PS to the PL
  - 64 bits of GPIO
  - 4 GP AXI Ports
  - 4 HP AXI Ports
  - 1 ACP Port

Just talked about this



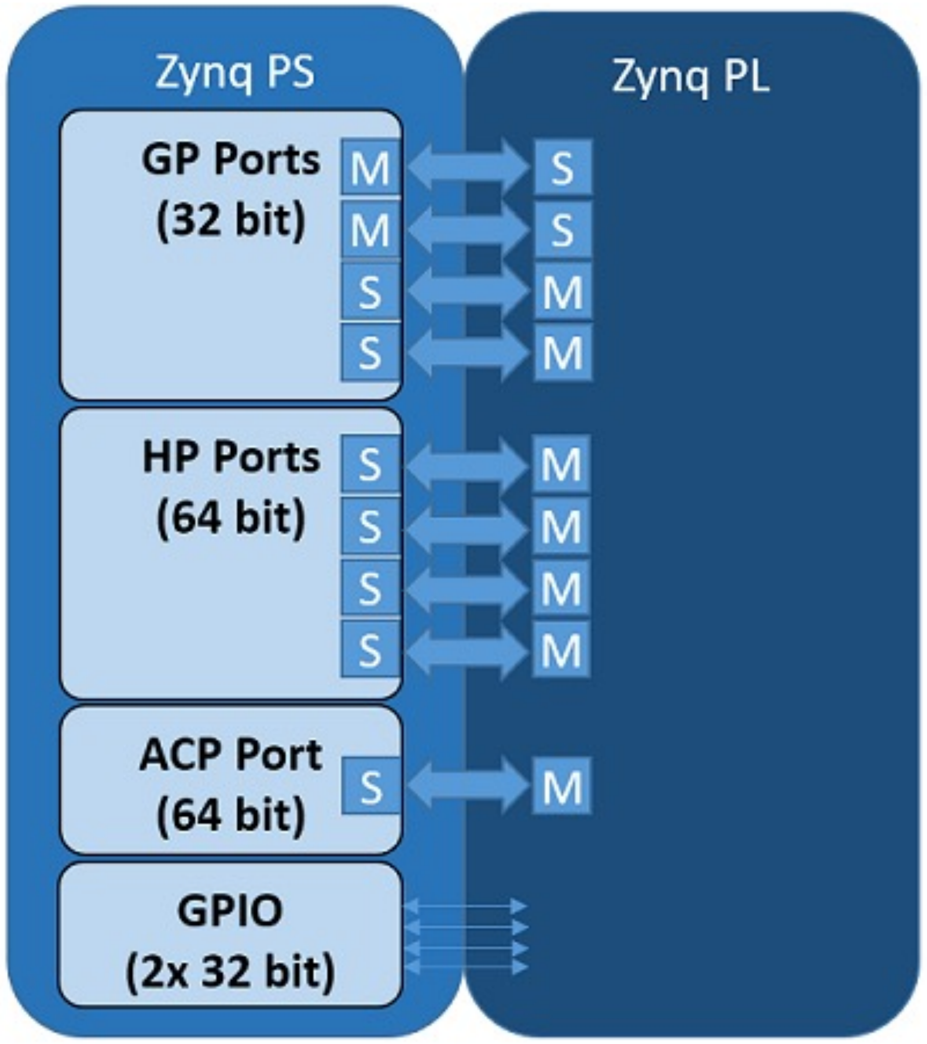
https://pynq.readthedocs.io/en/v2.3/overlay_design_methodology/pspl_interface.html

Master/Slave Terminology

- I’ve been a big fan of moving away from this terminology.
- For SPI, for example, instead of MOSI/MISO, do COPI/CIPO (controller/peripheral), etc…
- **However**, ***all*** of the AMD/Xilinx, use Master/Slave and ***everything*** has that M’s and S’s prepended, appended, etc..
- I’m going to just use their nomenclature so we don’t have to constantly be mapping between alternate names.

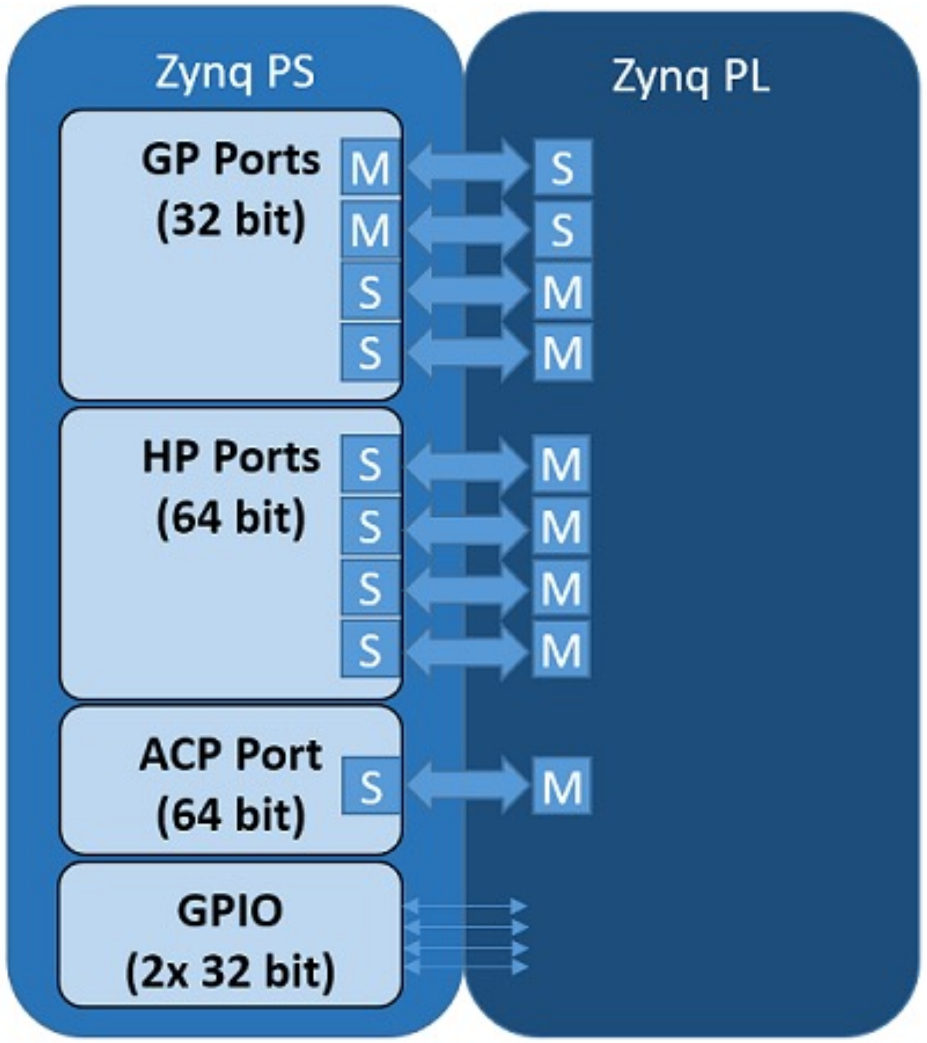
AXI Ports

- Parallel Busses of two different flavors that allow us to pretty quickly transfer data between the Processing System and the FPGA section using shared registers and some other stuff



ACP Port

- Accelerator Coherency Port
- 64-bit wide bus that can transfer data from very quickly from PL fabric

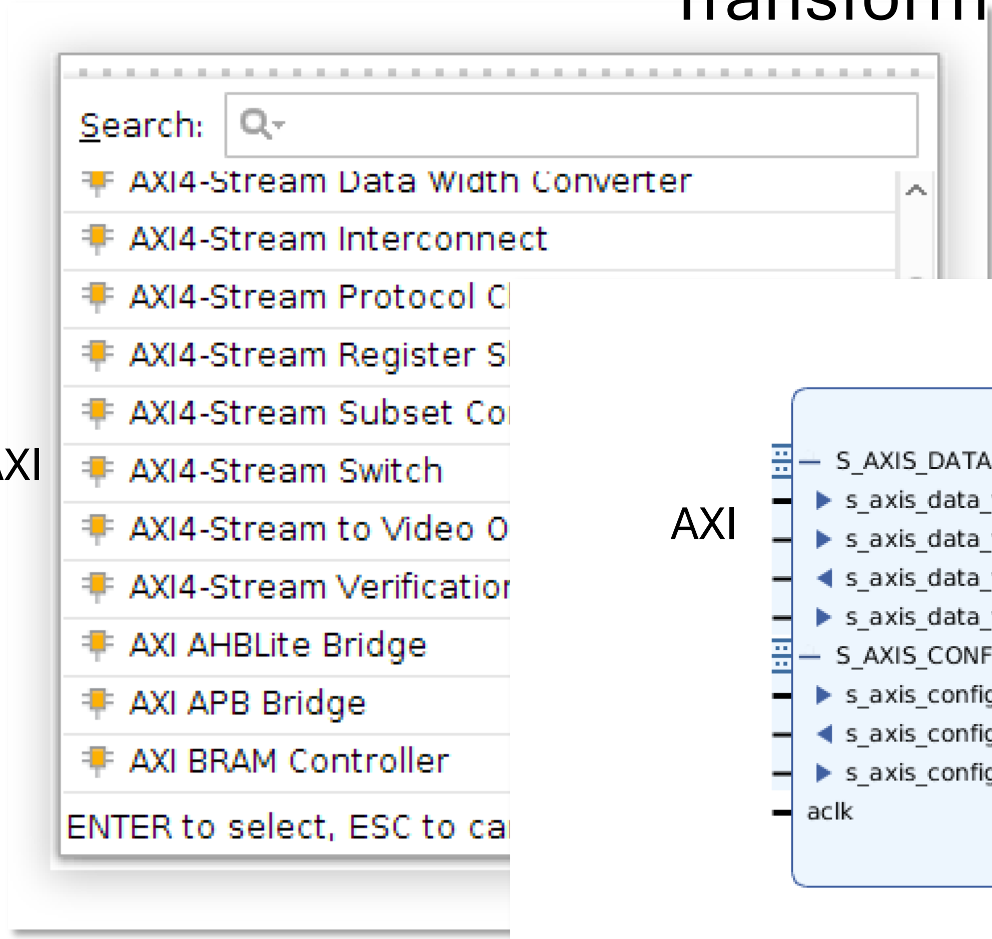


AXI Everywhere

- There's lot of neat IP we can work with....if you wanted to implement a hardware accelerated Fast Fourier Transform you totally can...

AXI

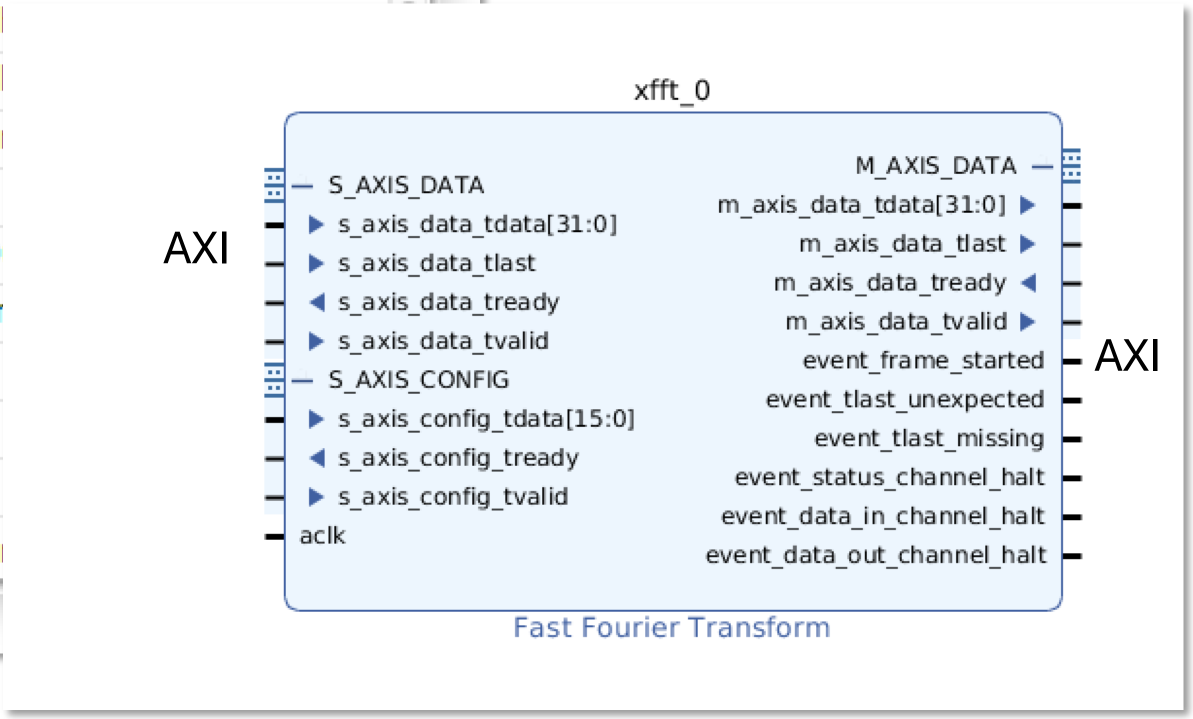
AXI


Search:

- AXI4-Stream Data Width Converter
- AXI4-Stream Interconnect
- AXI4-Stream Protocol C
- AXI4-Stream Register S
- AXI4-Stream Subset Co
- AXI4-Stream Switch
- AXI4-Stream to Video O
- AXI4-Stream Verificatio
- AXI AHBLite Bridge
- AXI APB Bridge
- AXI BRAM Controller

ENTER to select, ESC to ca

AXI



AXI

Advanced Microcontroller Bus Architecture (AMBA)

- Version 1 released in 1996 by ARM
- 2003 saw release of **A**dvanced e**X**tensible **I**nterface (AXI3)
- 2011 saw release of AXI4
- There are no royalties affiliated with AMBA/AXI so they're used a lot.
- It is a general, flexible, and relatively free* communication protocol for development

https://en.wikipedia.org/wiki/Advanced_Microcontroller_Bus_Architecture

Three General Flavors of AXI4

- **AXI4 (Full AXI):** For memory-mapped links. Provides highest performance.
  1. Address is supplied
  2. Then a data burst transfer of up to 256 data words
- **AXI4 Lite:** A memory-mapped simplified link supporting only one data transfer per connection (no bursts). (also restricted to 32 bit addr/data)
  1. Address is supplied
  2. One data transfer
- **AXI4 Stream:** Meant high-speed streaming data
  - Can do burst transfers of unrestricted size
  - No addressing
  - Meant to stream data from one device to another quickly on its own direct connection

From the Zynq Book

Memory Map?

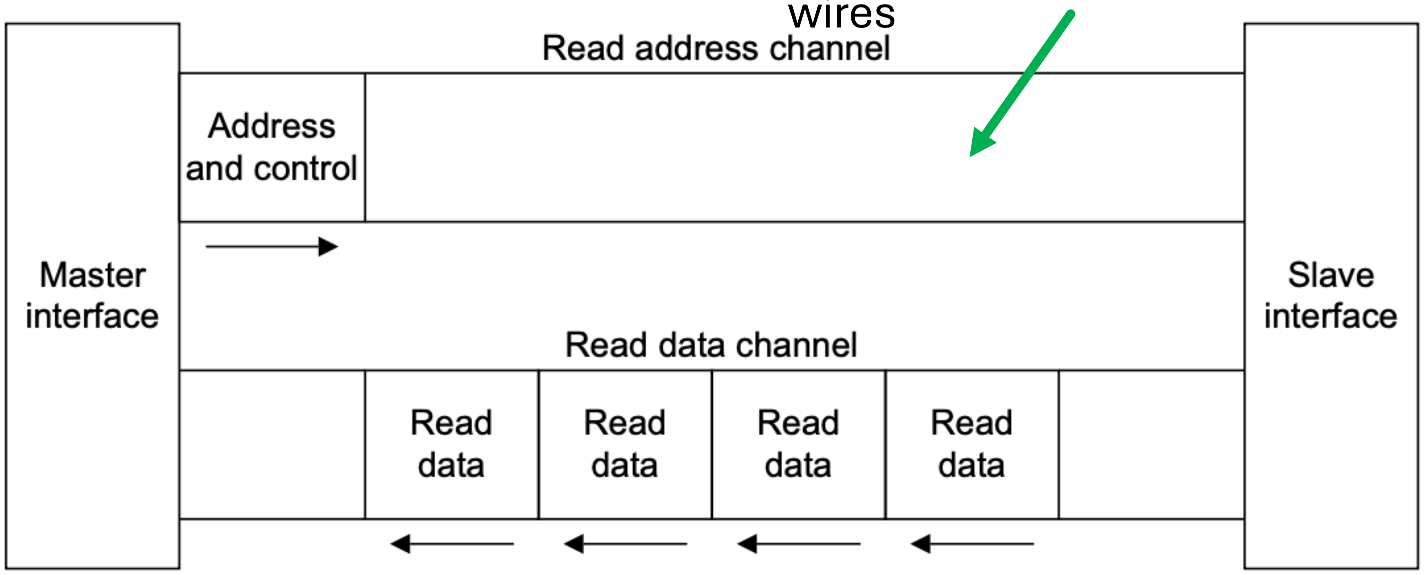
- Memory mapped means an address is specified within the transaction by the master (read or write). This corresponds to an address in the system memory space.
- For **AXI4-Lite**, which supports a single data transfer per transaction, data is then written to, or read from, the specified address
- For **Full-AXI4** sending a burst, the address specified is for the first data word to be transferred, and the slave must then calculate the addresses for the data words that follow.
- **AXI-Stream** has no addressing so no memory mapping

AXI Idea

- Communication between two devices (Master and Slave) is carried out over multiple assigned “channels”
- Each channel has its own collection of wires which convey data, signals, etc.
- The channels can work somewhat independently, however in practice what one channel does is often the result of what a different one did previously
- Five Types of Channels (may have all or a subset):
    - Read Address: ”AR” channel
    - Read Data: “R” channel
    - Write Address: “AW” channel
    - Write Data: “W” channel
    - Write Response: “B” channel

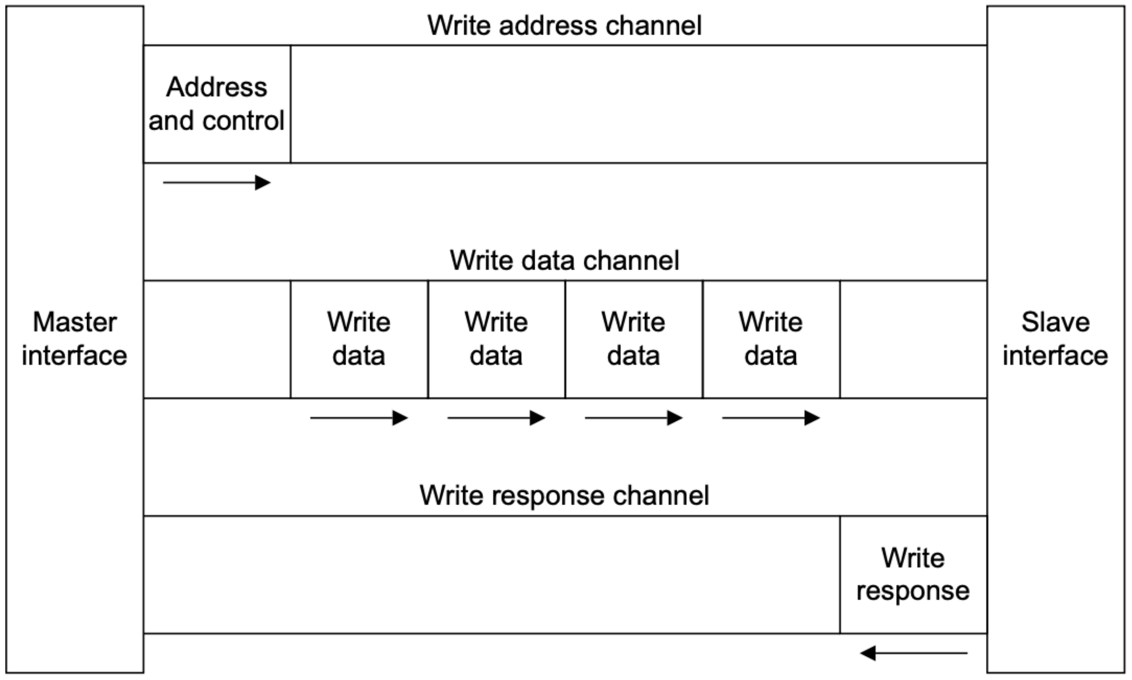
Read Wiring

Generalized collection of wires "Channel". Will contain numerous wires



Master initiates communication, Slave responds

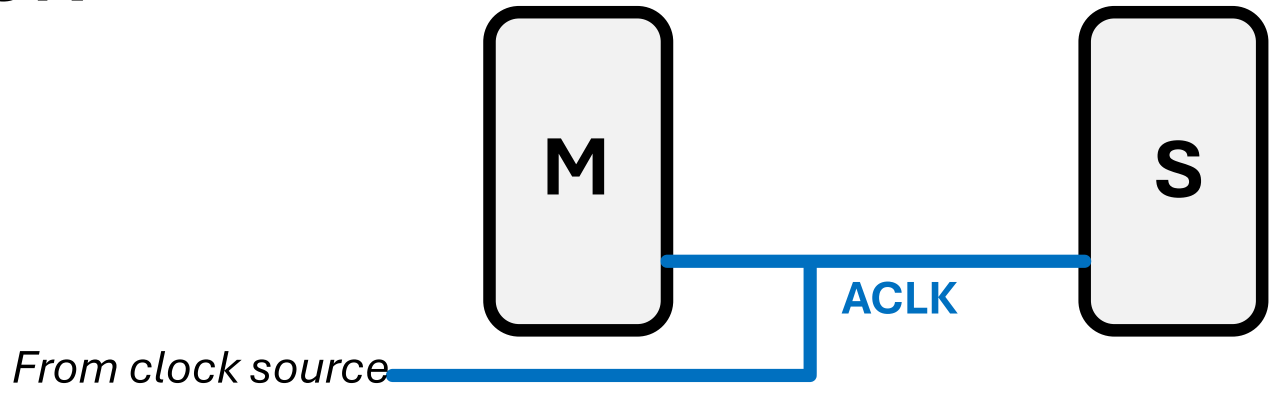
Write Wiring



Within Each Channel are wires:

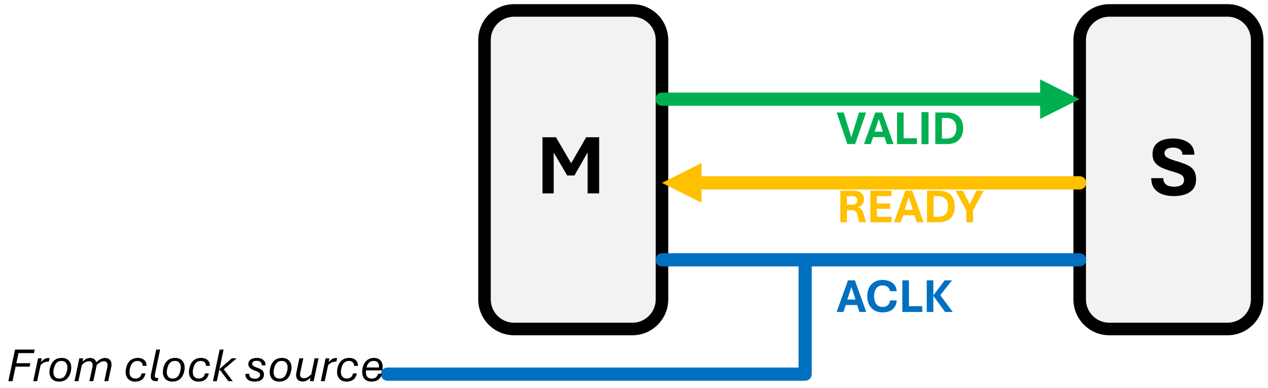
- These wires serve specific purposes.
- Some are universal to all channels, and others are specific

AXI Clock



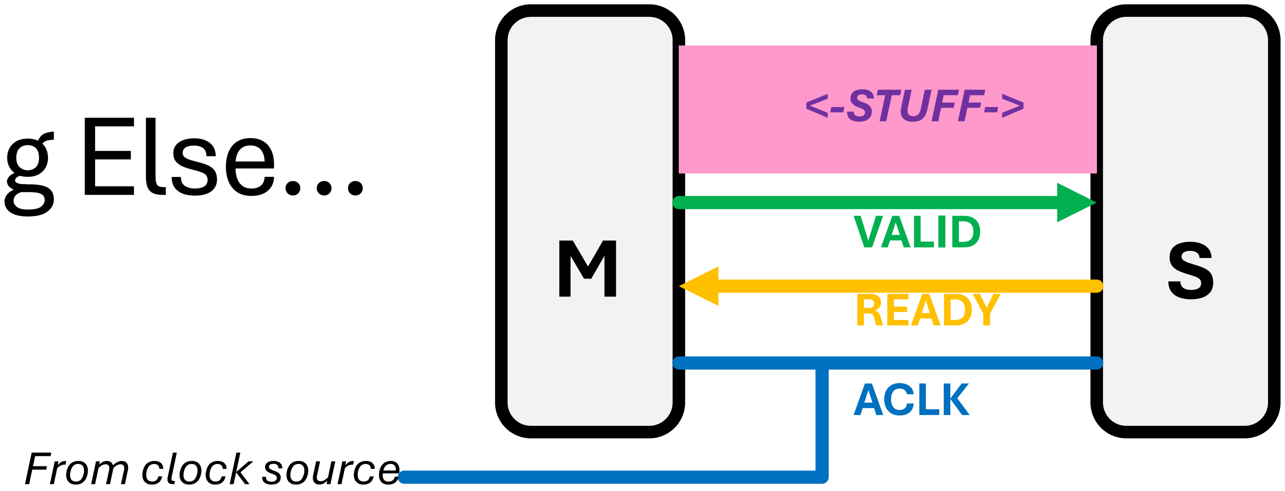
- Everything in system will run off of AXI clock usually called **ACLK** in documentation
- No combinatorial paths between inputs and outputs. Everything must be registered.
- All signals are sampled **on rising edge**
- AXI modules should also have Reset pins. AXI work ACTIVE LOW so the Reset pin is usually called **ARSTn** or **ARESETn**

Valid and Ready



- All of AXI uses the same handshake procedure:
- The source of a data generates a **VALID** signal
- The destination generates a **READY** signal
- Transfer of data only occurs when both are high
- Both Master and Slave Devices can therefore control the flow of their data as needed

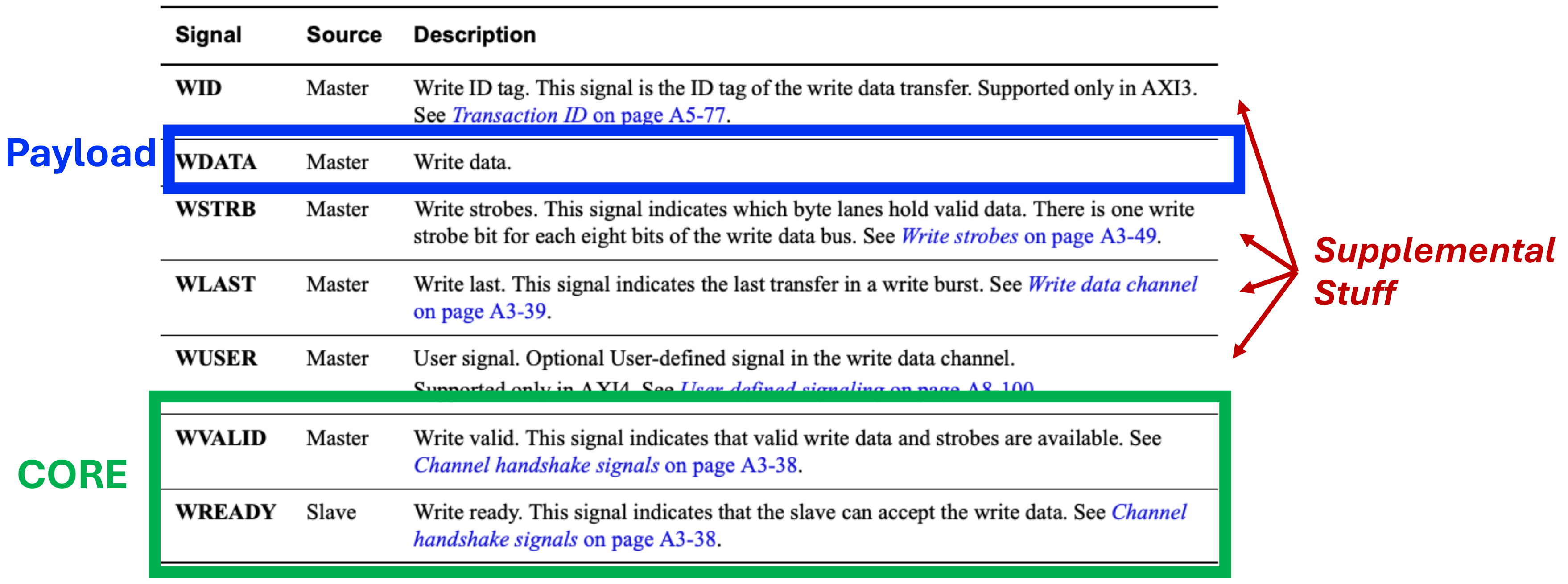
Everything Else…



- Everything else is information and depends on what is needed in situation. Could be:
  - Address
  - Data
  - Other specialized wires like:
    - STRB (used to specify which bytes in current data step are valid, sent by Master along with data payload to Slave)
    - RESP (sort of like a status
    - LAST (sent to indicate the final data clock cycle of data in a burst)

Each channel has its own subset of “stuff” that goes along with those core signals shared by all

For example, the Write Data Channel (”W” channel)


Payload
Supplemental Stuff
CORE

Signal	Source	Description
WID	Master	Write ID tag. This signal is the ID tag of the write data transfer. Supported only in AXI3. See *Transaction ID* on page A5-77.
WDATA	Master	Write data.
WSTRB	Master	Write strobes. This signal indicates which byte lanes hold valid data. There is one write strobe bit for each eight bits of the write data bus. See *Write strobes* on page A3-49.
WLAST	Master	Write last. This signal indicates the last transfer in a write burst. See *Write data channel* on page A3-39.
WUSER	Master	User signal. Optional User-defined signal in the write data channel. Supported only in AXI4. See *User-defined signaling* on page A8-100.
WVALID	Master	Write valid. This signal indicates that valid write data and strobes are available. See *Channel handshake signals* on page A3-38.
WREADY	Slave	Write ready. This signal indicates that the slave can accept the write data. See *Channel handshake signals* on page A3-38.

The Read Data Channel:

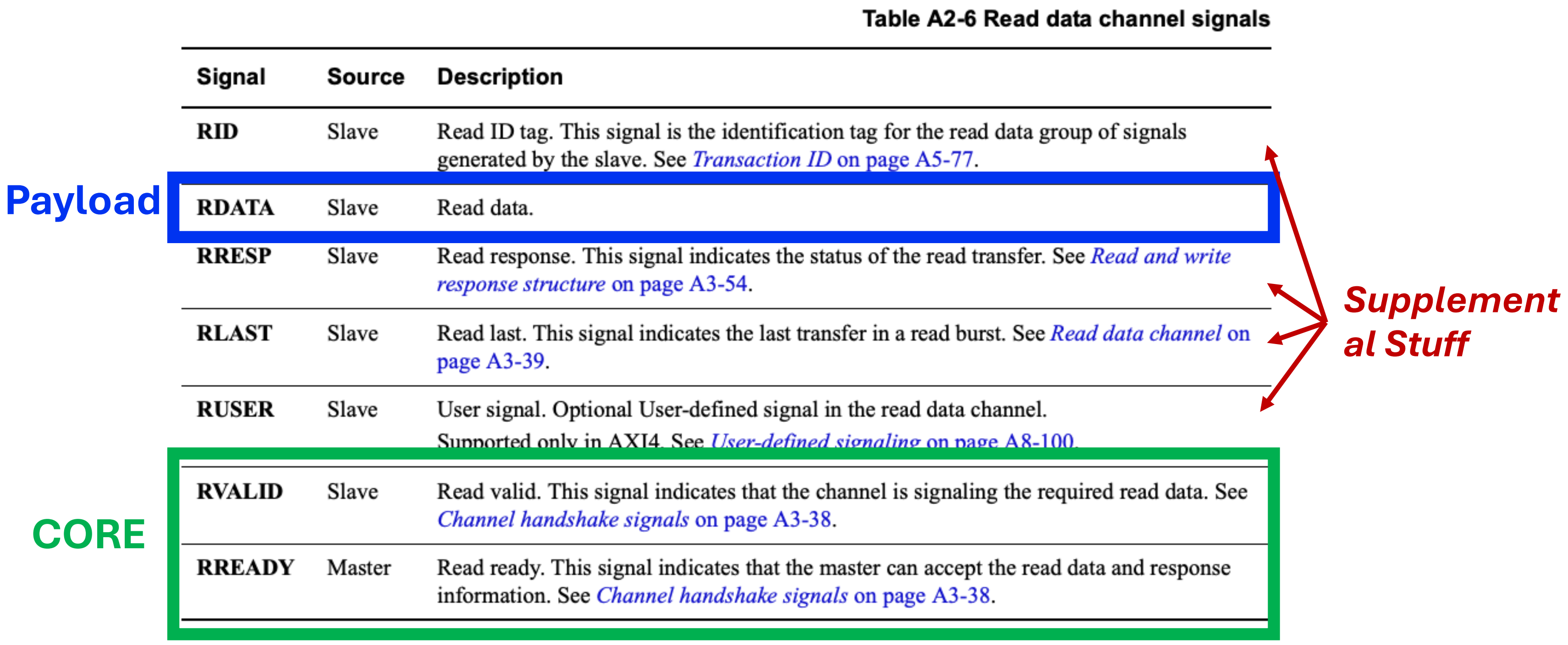

Payload
Supplemental Stuff
CORE

Table A2-6 Read data channel signals

Signal	Source	Description
RID	Slave	Read ID tag. This signal is the identification tag for the read data group of signals generated by the slave. See *Transaction ID* on page A5-77.
RDATA	Slave	Read data.
RRESP	Slave	Read response. This signal indicates the status of the read transfer. See *Read and write response structure* on page A3-54.
RLAST	Slave	Read last. This signal indicates the last transfer in a read burst. See *Read data channel* on page A3-39.
RUSER	Slave	User signal. Optional User-defined signal in the read data channel. Supported only in AXI4. See *User-defined signaling* on page A8-100.
RVALID	Slave	Read valid. This signal indicates that the channel is signaling the required read data. See *Channel handshake signals* on page A3-38.
RREADY	Master	Read ready. This signal indicates that the master can accept the read data and response information. See *Channel handshake signals* on page A3-38.

Read Address Chanel

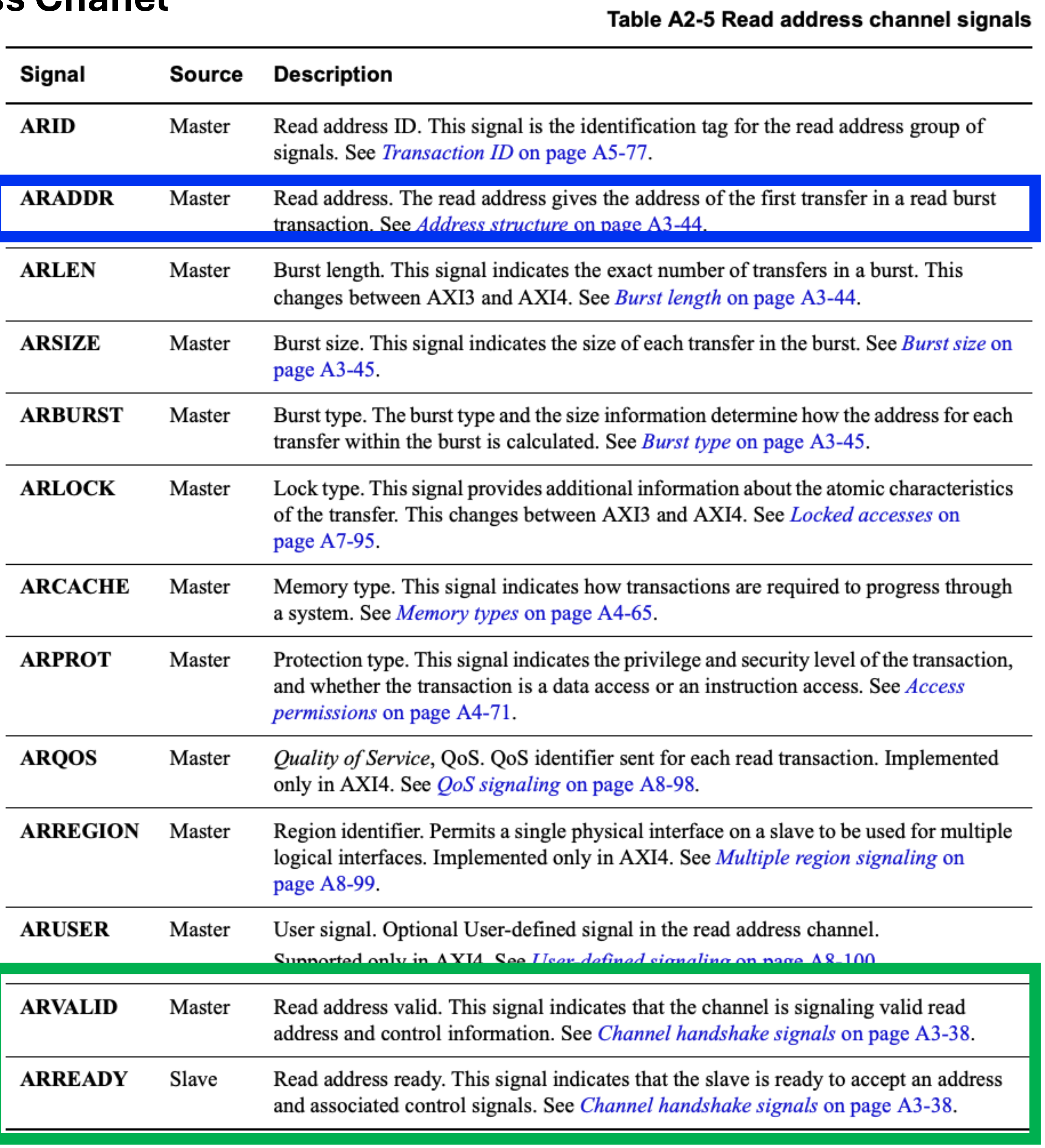


Table A2-5 Read address channel signals

Signal	Source	Description
ARID	Master	Read address ID. This signal is the identification tag for the read address group of signals. See *Transaction ID* on page A5-77.
ARADDR	Master	Read address. The read address gives the address of the first transfer in a read burst transaction. See *Address structure* on page A3-44.
ARLEN	Master	Burst length. This signal indicates the exact number of transfers in a burst. This changes between AXI3 and AXI4. See *Burst length* on page A3-44.
ARSIZE	Master	Burst size. This signal indicates the size of each transfer in the burst. See *Burst size* on page A3-45.
ARBURST	Master	Burst type. The burst type and the size information determine how the address for each transfer within the burst is calculated. See *Burst type* on page A3-45.
ARLOCK	Master	Lock type. This signal provides additional information about the atomic characteristics of the transfer. This changes between AXI3 and AXI4. See *Locked accesses* on page A7-95.
ARCACHE	Master	Memory type. This signal indicates how transactions are required to progress through a system. See *Memory types* on page A4-65.
ARPROT	Master	Protection type. This signal indicates the privilege and security level of the transaction, and whether the transaction is a data access or an instruction access. See *Access permissions* on page A4-71.
ARQOS	Master	*Quality of Service*, QoS. QoS identifier sent for each read transaction. Implemented only in AXI4. See *QoS signaling* on page A8-98.
ARREGION	Master	Region identifier. Permits a single physical interface on a slave to be used for multiple logical interfaces. Implemented only in AXI4. See *Multiple region signaling* on page A8-99.
ARUSER	Master	User signal. Optional User-defined signal in the read address channel. Supported only in AXI4. See *User-defined signaling* on page A8-100.
ARVALID	Master	Read address valid. This signal indicates that the channel is signaling valid read address and control information. See *Channel handshake signals* on page A3-38.
ARREADY	Slave	Read address ready. This signal indicates that the slave is ready to accept an address and associated control signals. See *Channel handshake signals* on page A3-38.

Payload

CORE

Write Response

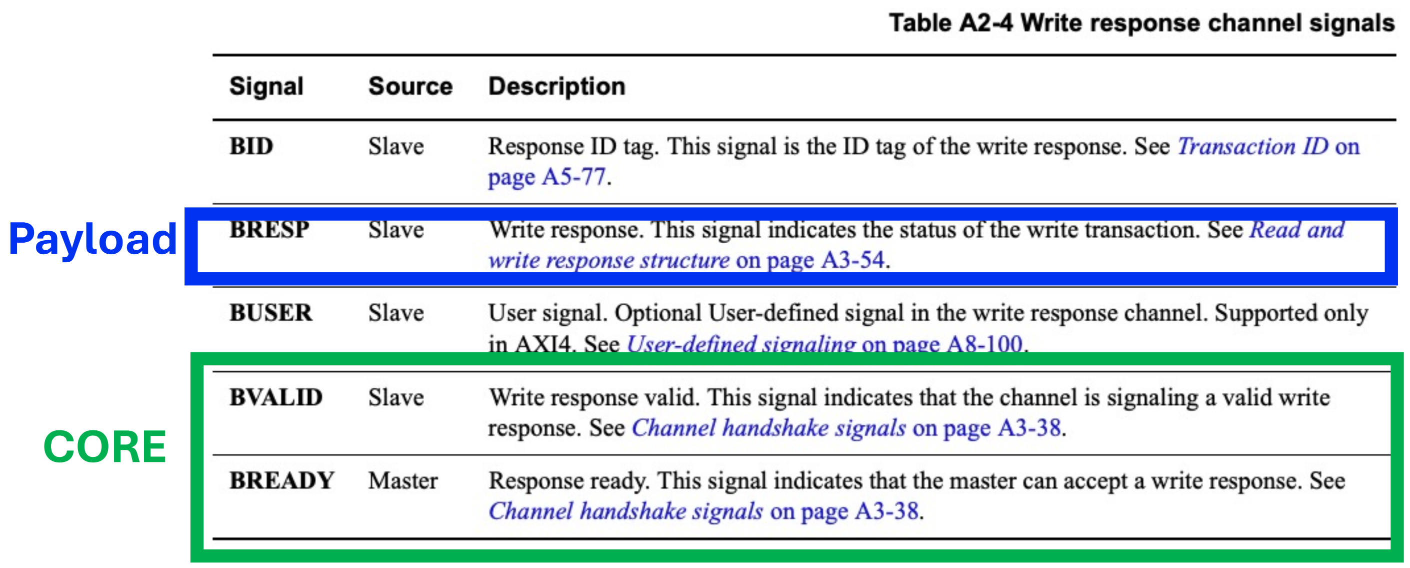

Payload
CORE

Table A2-4 Write response channel signals

Signal	Source	Description
BID	Slave	Response ID tag. This signal is the ID tag of the write response. See *Transaction ID* on page A5-77.
BRESP	Slave	Write response. This signal indicates the status of the write transaction. See *Read and write response structure* on page A3-54.
BUSER	Slave	User signal. Optional User-defined signal in the write response channel. Supported only in AXI4. See *User-defined signaling* on page A8-100.
BVALID	Slave	Write response valid. This signal indicates that the channel is signaling a valid write response. See *Channel handshake signals* on page A3-38.
BREADY	Master	Response ready. This signal indicates that the master can accept a write response. See *Channel handshake signals* on page A3-38.

Write Address Channel

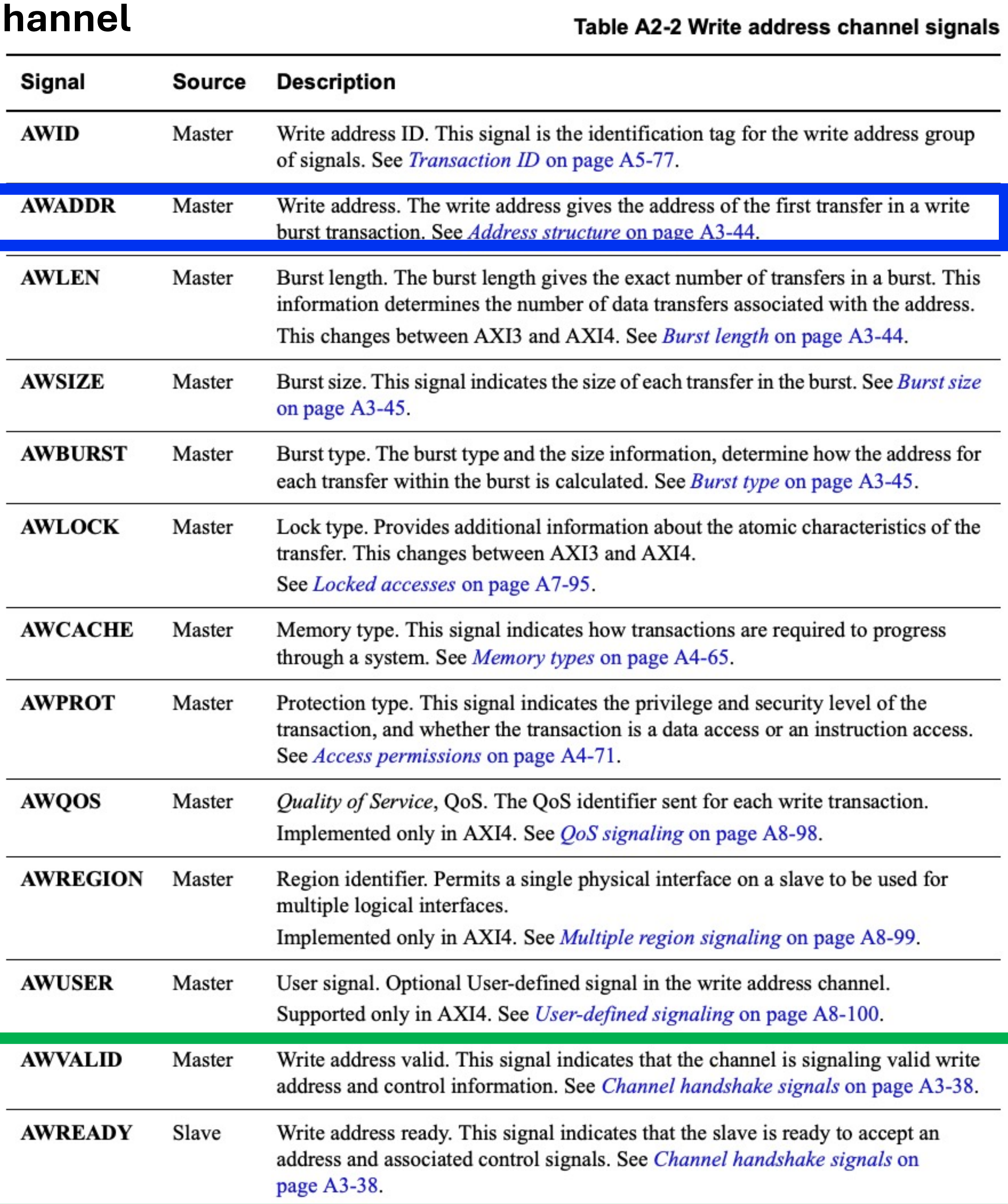


Table A2-2 Write address channel signals

Signal	Source	Description
AWID	Master	Write address ID. This signal is the identification tag for the write address group of signals. See *Transaction ID* on page A5-77.
AWADDR	Master	Write address. The write address gives the address of the first transfer in a write burst transaction. See *Address structure* on page A3-44.
AWLEN	Master	Burst length. The burst length gives the exact number of transfers in a burst. This information determines the number of data transfers associated with the address. This changes between AXI3 and AXI4. See *Burst length* on page A3-44.
AWSIZE	Master	Burst size. This signal indicates the size of each transfer in the burst. See *Burst size* on page A3-45.
AWBURST	Master	Burst type. The burst type and the size information, determine how the address for each transfer within the burst is calculated. See *Burst type* on page A3-45.
AWLOCK	Master	Lock type. Provides additional information about the atomic characteristics of the transfer. This changes between AXI3 and AXI4. See *Locked accesses* on page A7-95.
AWCACHE	Master	Memory type. This signal indicates how transactions are required to progress through a system. See *Memory types* on page A4-65.
AWPROT	Master	Protection type. This signal indicates the privilege and security level of the transaction, and whether the transaction is a data access or an instruction access. See *Access permissions* on page A4-71.
AWQOS	Master	*Quality of Service*, QoS. The QoS identifier sent for each write transaction. Implemented only in AXI4. See *QoS signaling* on page A8-98.
AWREGION	Master	Region identifier. Permits a single physical interface on a slave to be used for multiple logical interfaces. Implemented only in AXI4. See *Multiple region signaling* on page A8-99.
AWUSER	Master	User signal. Optional User-defined signal in the write address channel. Supported only in AXI4. See *User-defined signaling* on page A8-100.
AWVALID	Master	Write address valid. This signal indicates that the channel is signaling valid write address and control information. See *Channel handshake signals* on page A3-38.
AWREADY	Slave	Write address ready. This signal indicates that the slave is ready to accept an address and associated control signals. See *Channel handshake signals* on page A3-38.

Payload

CORE

Generalized Transaction

- All Channel Interactions follow same high-level structure

Table A3-1 Transaction channel handshake pairs

Transaction channel	Handshake pair
Write address channel	**AWVALID**, **AWREADY**
Write data channel	**WVALID**, **WREADY**
Write response channel	**BVALID**, **BREADY**
Read address channel	**ARVALID**, **ARREADY**
Read data channel	**RVALID**, **RREADY**

Sending One ”beat” of data (one clock-cycle of data)

Keep in mind this could be 64 parallel wires of 1’s and 0’s of info or 8 bytes for example…
Or it could be something else

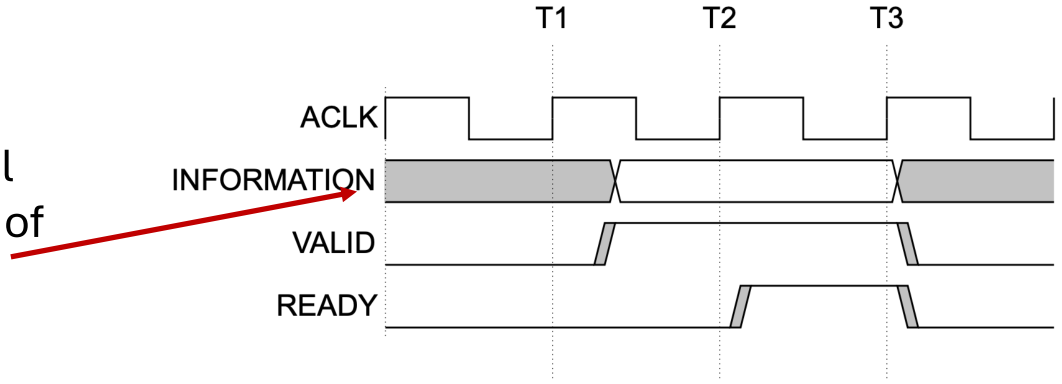


Figure A3-2 VALID before READY handshake

Generalized Transaction

- All Channel Interactions follow same high-level structure

Table A3-1 Transaction channel handshake pairs

Transaction channel	Handshake pair
Write address channel	**AWVALID, AWREADY**
Write data channel	**WVALID, WREADY**
Write response channel	**BVALID, BREADY**
Read address channel	**ARVALID, ARREADY**
Read data channel	**RVALID, RREADY**

Sending One ”beat” of data (one clock-cycle of data)

Keep in mind this could be 64 parallel wires of 1’s and 0’s of info or 8 bytes for example…
Or it could be something else

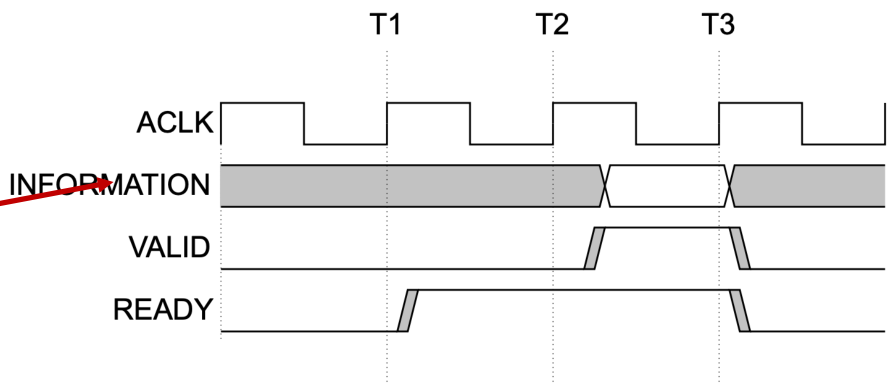


Figure A3-3 READY before VALID handshake

Generalized Transaction

Table A3-1 Transaction channel handshake pairs

Transaction channel	Handshake pair
Write address channel	**AWVALID, AWREADY**
Write data channel	**WVALID, WREADY**
Write response channel	**BVALID, BREADY**
Read address channel	**ARVALID, ARREADY**
Read data channel	**RVALID, RREADY**

- All Channel Interactions follow same high-level structure

Sending One "beat" of data (one clock-cycle of data)

Keep in mind this could be 64 parallel wires of 1's and 0's of info or 8 bytes for example...
Or it could be something else

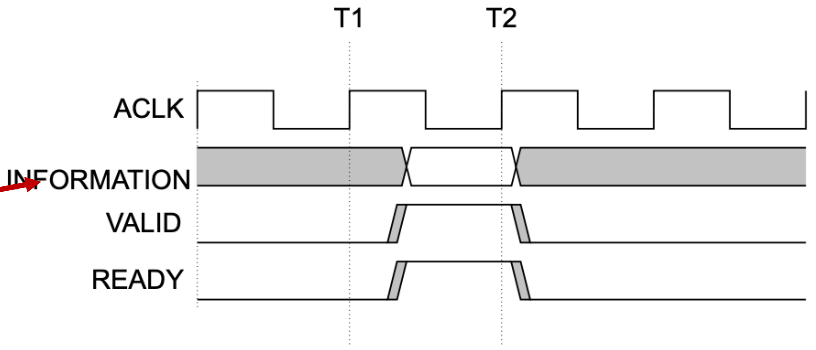


Figure A3-4 VALID with READY handshake

Other Things to Keep in Mind

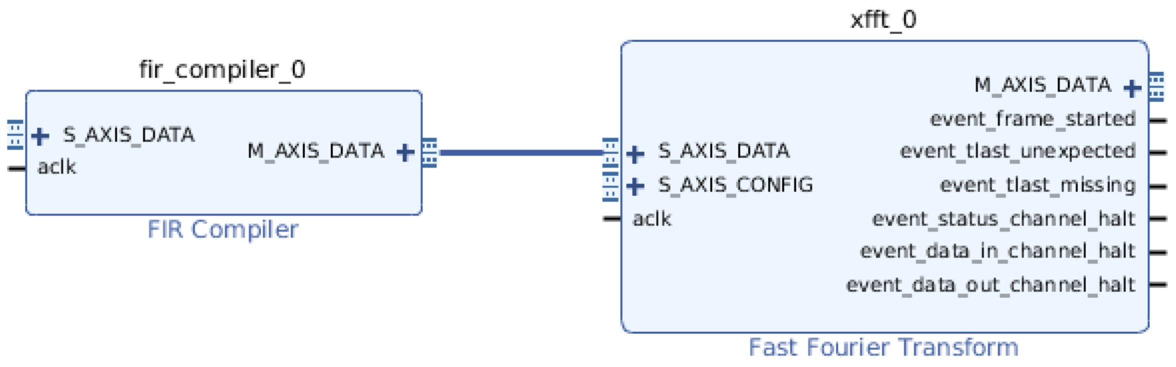
- the VALID signal of the AXI interface sending information must not be dependent on the READY signal of the AXI interface receiving that information
- an AXI interface that is receiving information can wait until it detects a VALID signal before it asserts its corresponding READY signal.
- Fail to Follow these rules and could have devices wait infinitely.
  - Like when two people keep going "no, after you at a door"

And Up to All Five AXI channels can come from one device

- While operating independently at their individual transaction level, they can then report to the larger module to provide overall interfaces
- Example:
  - The slave device receives address on write channel address
  - The write data channel then becomes active and knows where to point incoming data
  - The response channel then opens and does its thing
  - And so on
- Hierarchy of Control/Design

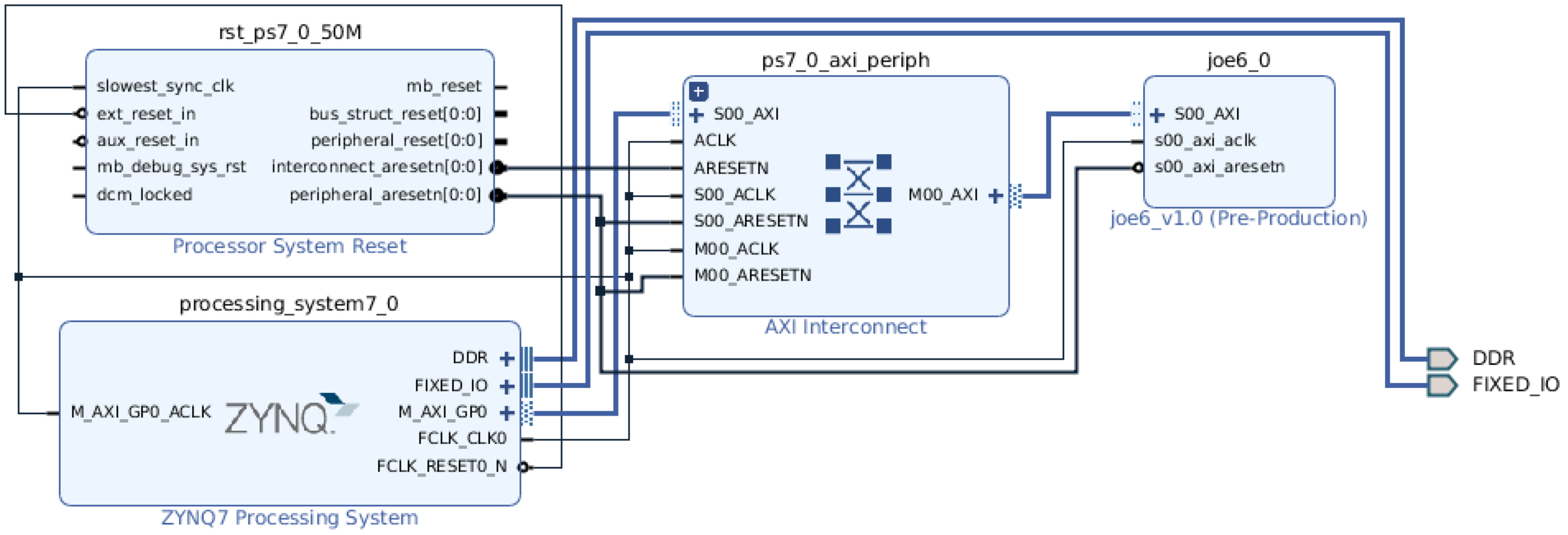
And you Can Use AXI to Interface with Tons of things!

Connecting a FIR (from a Xilinx IP) to the FFT module



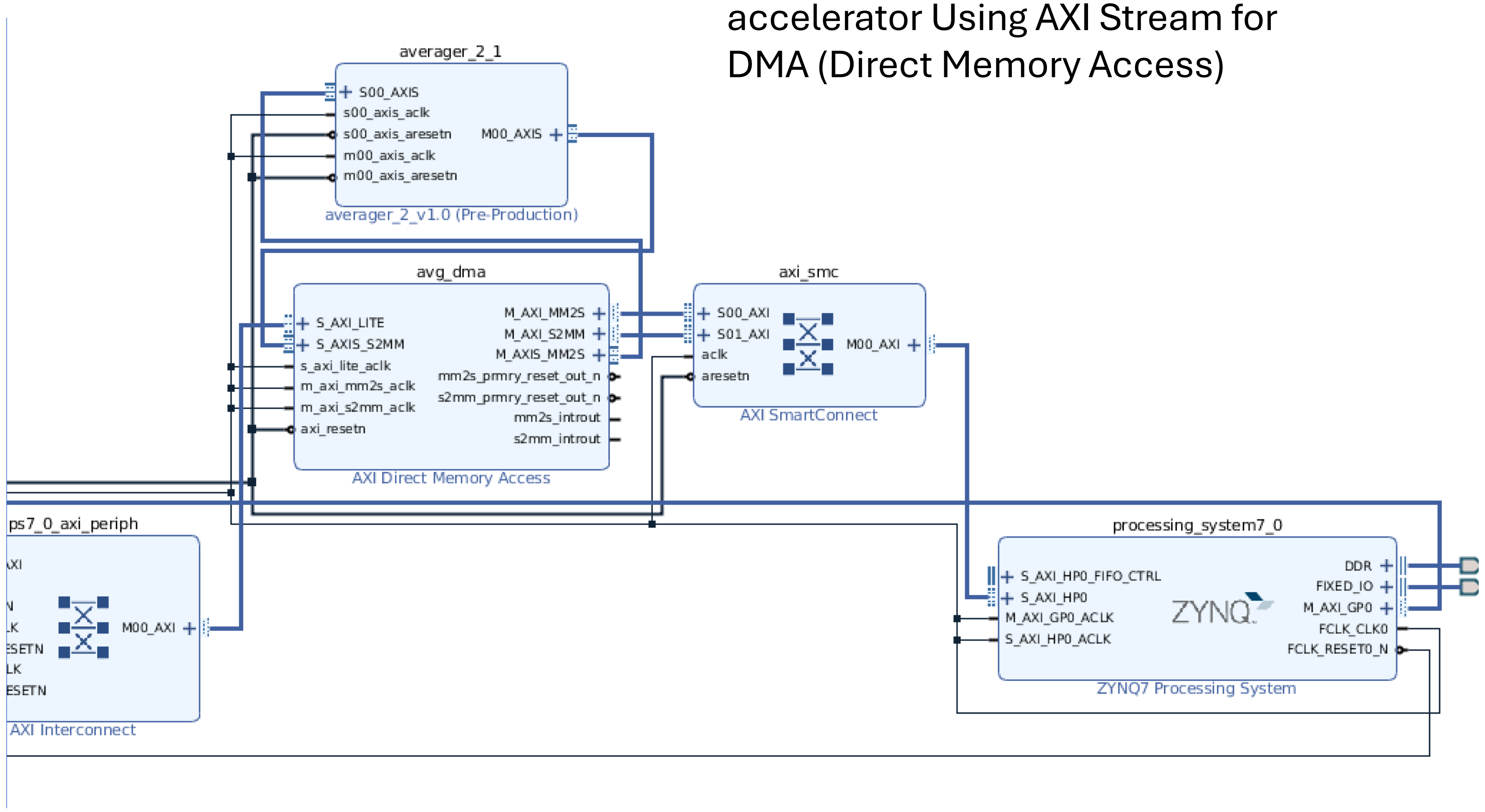
And you Can Use AXI to Interface with Tons of things!

Creating a AXI-controlled joe6 module that I can then call from Python

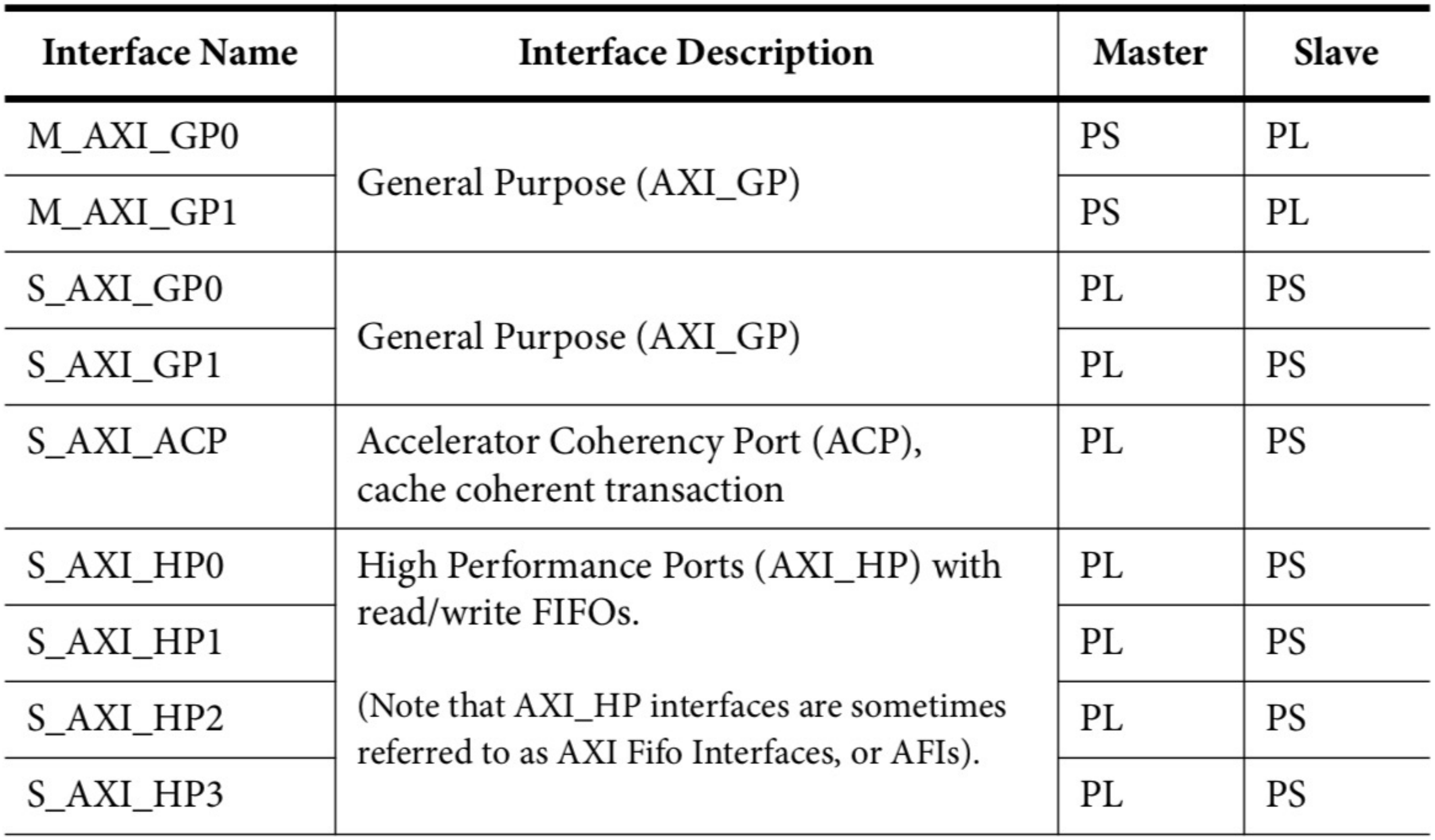


And you Can Use AXI to Interface with Tons of things!

A running-average hardware accelerator Using AXI Stream for DMA (Direct Memory Access)



The AXI Interfaces on the Zynq Enable PS to PL communication effectively



Interface Name	Interface Description	Master	Slave
M_AXI_GP0	General Purpose (AXI_GP)	PS	PL
M_AXI_GP1		PS	PL
S_AXI_GP0	General Purpose (AXI_GP)	PL	PS
S_AXI_GP1		PL	PS
S_AXI_ACP	Accelerator Coherency Port (ACP), cache coherent transaction	PL	PS
S_AXI_HP0	High Performance Ports (AXI_HP) with read/write FIFOs. (Note that AXI_HP interfaces are sometimes referred to as AXI Fifo Interfaces, or AFIs).	PL	PS
S_AXI_HP1		PL	PS
S_AXI_HP2		PL	PS
S_AXI_HP3		PL	PS

Master/Slave refers to who controls/initiates comms on that bus that bus

From Zynq Book

General Purpose/Performance "GP" AXI Ports

- 32 bits in size
- Maximum flexibility
- Allow register access from:
  - PS to PL
  - PL to PS

High Performance “HP” AXI Ports

- Can be 32 or 64 bits wide (or variable between, but avoid)
- Maximum bandwidth access to external memory and on-chip-memory (OCM)
- When use all four HP ports at 64 bits, you can outpace ability to write to DDR and OCM bandwidths!
  - HP Ports : 4 * 64 bits * 150 MHz * 2 = **9.6 GByte/sec**
  - external DDR: 1 * 32 bits * 1066 MHz * 2 = **4.3 GByte/sec**
  - OCM : 64 bits * 222 MHz * 2 = **3.5 GByte/sec**
- Optimized for large burst lengths

Taken from ECE699 lec 6 notes gm.edu

How it is Laid Out

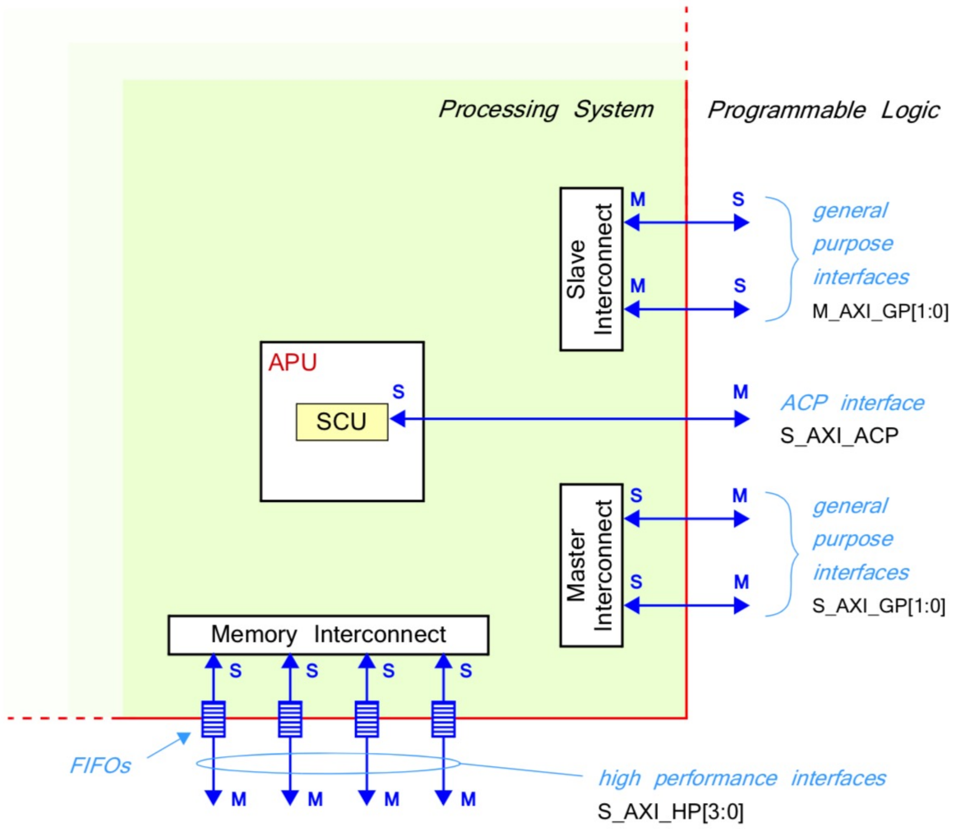


Figure 2.9: The structure of AXI interconnects and interfaces connecting the PS and PL

From The Zynq Book

Complexity

- In terms of wires and options, Full-AXI is the most complex
- AXI-LITE has a lot less options (single data beat so all the supplemental stuff that specifies burst characteristics gets skipped)
- AXI-STREAM has even less...basically a high-speed write channel (Few options), but often needs that extra TLAST signal

$$\begin{tikzcd}
\text{Full-AXI4} \arrow[d] \\
\text{AXI-LITE} \arrow[d] \\
\text{AXI-STREAM}
\end{tikzcd}$$

Sources

This is the thing right here…the spec sheet/manual is surprisingly good!!

- **“AMBA® AXITM and ACETM Protocol Specification”,** ARM 2011
- **“The Zynq Book”, L.H. Crockett, R.A. Elliot, M.A. Enderwitz, and R.W. Stewart, University of Glasgow**
- **“Building Zynq Accelerators with Vivado High Level Synthesis” Xilinx Technical Note**
- **Some material from ECE699 Spring 2016 https://ece.gmu.edu/coursewebpages/ECE/ECE699_SW_HW/S16/**

Crack open the AXI spec sheet with a few data sheets for some Xilinx IP cores (like the CORDIC, FFT, etc…) and you should be able to start making sense of it.