6.S965
Digital Systems Laboratory II

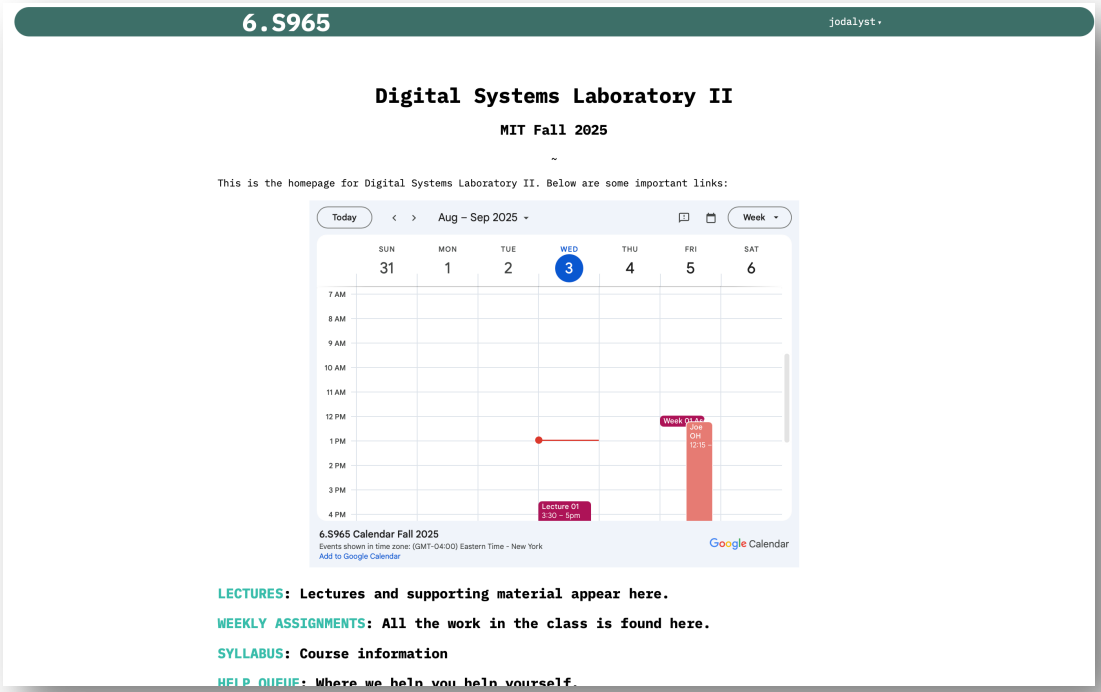
Lecture 1:
Introduction

Course Overview

- Instructor: Joe Steinmeyer (jodalyst)
- TA: Oliver Trevor (olt)
- Prerequisites: 6.205
- Units: 2-8-2
- Lecture 3:30-5:00(ish)**** MW in 36-112
- Lab is rolling (similar to 6.205).
- Site: **https://rfsoc.mit.edu/6S965/F25**
- Calendar on front of site
- Grad Course which "Satisfies: II, DLAB, 6-2 PLAB, AAGS; EE Track: Architecture, Hardware Design, Embedded systems; Concentration subject in either Circuits or Computer Systems"

Course Calendar

- When Oliver and I are in lab will show up on calendar.
- Also due dates, etc…



Course Content

- Keep Studying FPGAs and associated technologies including:
  - Systems on Chips (SoCs, RFSoCs)
  - Common Peripherals (DRAM, ADCs, DACs)
- Study some signal processing concepts and RF-related stuff
- Investigate and study modern verification techniques:
  - Cocotb with more rigor
  - Talk about verification as a field
- Books: We'll use several:
  - *Software Defined Radio with Zynq Ultrascale+ RFSoC* (linked on site)
  - Few others as needed.

Schedule

- Sept 3 to Oct 24: Weekly assignments and lecture/recitations
  - Lectures Monday/Wednesday 3:30-5:00*
  - Weekly assignments: Friday-to-Friday release-due cycle
- Oct 24 to Dec 10: Final Project (ideally in teams):
  - No real fixed schedule. I'll meet with you and your team periodically to check in
  - I'll want a written up report and video demonstration (live demonstration would be even better)

Weekly Assignments

- Some Lab Work, likely and largely using Zynq 7000 and Ultrascale+ SoCs and RFSoCs
- Some Modeling/Simulation/Verification exercises
- Assignments will come out on Fridays and be due the following Fridays. (First one on Sept 5 and due on Sept 12)
- These are not “catsoop” questions. These will be some guided labs/things to design, write, and code up. When done you will submit them on the site via upload (code, mini/writeup, jupyter notebook, maybe video) and I will grade.

Lab

- We don’t have enough boards for everyone to have one. They stay in lab and you come in and use them. The lab machines also have the special licenses for
- The back area of the 6th floor I have made into the 6.S965 cocoon, though 6.205 may also sit there
- If you want to set up Vivado (2025.1) on your own laptop, that’s fine, and I can even give you a Enterprise-level license if you want, but you’re not obligated

Grading

- 50% Weekly Assignments from first half
- 50% Final Project

- This is a grad class. The work will be less than 6.205. I will expect you to need less guidance, though too.

- Do the work, be an active learner, try things, pursue a cool project, and your grade will be fine.

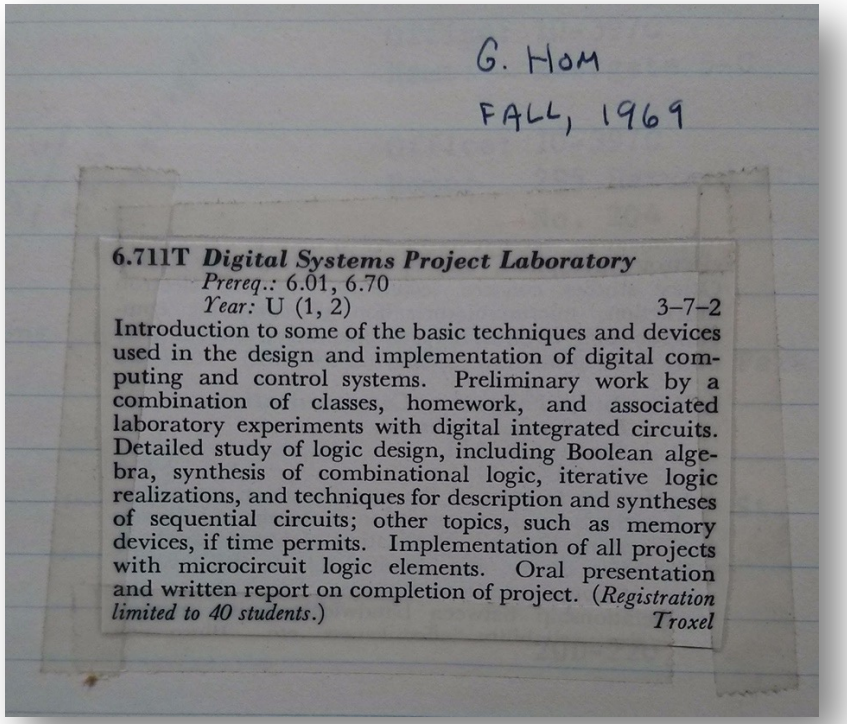
Final Projects

- Use any FPGA or SoC resources that you'd like
- We have some nice RFSoCs (15 of them), but it does not mean you need to use them if your project doesn't call for it.
  - I have a couple thousand dollars to spend on equipment for final projects to supplement the things if you go that route so if you need a better antenna, satellite dish, etc… we can make it happen
- I'd like us to put more effort put into verifying designs than in 6.205/111
- My hope is to then use final projects for better labs for next year. Wash/rinse/repeat

Motivation for Course

- In 1969, MIT had approximately one Digital Electronics Class (and occasionally a special subject in advanced topics like CMOS)

- The field has expanded immensely. Areas of work which were once a single unit in a class are now full career/research fields.


G. HOM
FALL, 1969

6.711T *Digital Systems Project Laboratory*
Prereq.: 6.01, 6.70
Year: U (1, 2) 3–7–2
Introduction to some of the basic techniques and devices used in the design and implementation of digital computing and control systems. Preliminary work by a combination of classes, homework, and associated laboratory experiments with digital integrated circuits. Detailed study of logic design, including Boolean algebra, synthesis of combinational logic, iterative logic realizations, and techniques for description and syntheses of sequential circuits; other topics, such as memory devices, if time permits. Implementation of all projects with microcircuit logic elements. Oral presentation and written report on completion of project. (*Registration limited to 40 students.*)
Troxel

2025 State of Courses (incomplete)

- 6.191: Computation Structures
- 6.192: Constructive Computer Architecture
- 6.590: Computer System Architecture
- 6.593: Hardware for Deep Learning
- 6.594: TinyML
- 6.595: Secure Hardware Design
- 6.205: Digital Systems Laboratory I
- 6.206: Microcontrollers
- 6.601: Analysis and Design of Digital Integrated Circuits
- 6.S965: Digital Systems II
- 6.S894: Accelerated Computing
- Etc…

It is an Exciting Time

- A lot of really cool ideas showing up in the hardware space now.
- Lots of cool tools, new designs, lots of money being spent, lots of impact.

Gaps

- The MIT course offering as it stands has some gaps:
- There are more and more heterogenous computing platforms getting developed:
  - GPUs
  - Accelerators
  - SoCs
  - AI-type engine/flavors
- Designs are getting so complex that new ways to test and verify them are getting developed and used

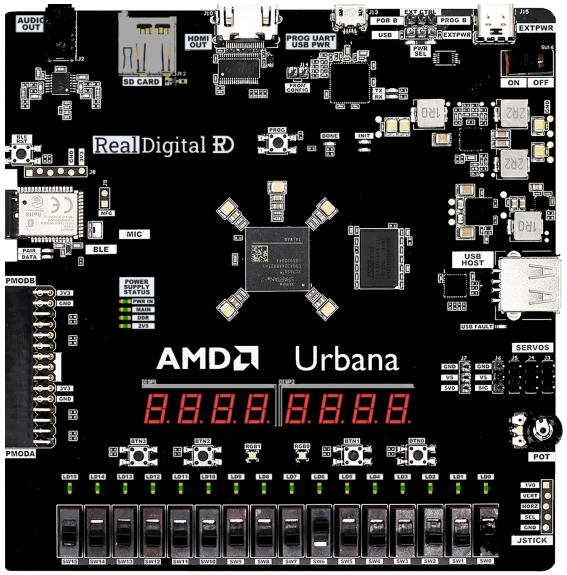
6.205/6.111

- Much of digital design ”in the lab” was concerned with wiring things up
- The proliferation of CPLDs and FPGAs throughout the 90’s early 2000’s automated much of the assembly portion
- The proliferation of HDLs and complicated build toolchains automated much of the algebraic factoring/design and reduction aspects of the field

Lab kit 1990 aka “digital death”

First FPGA Labkit at MIT ~2003

6.205 FPGA



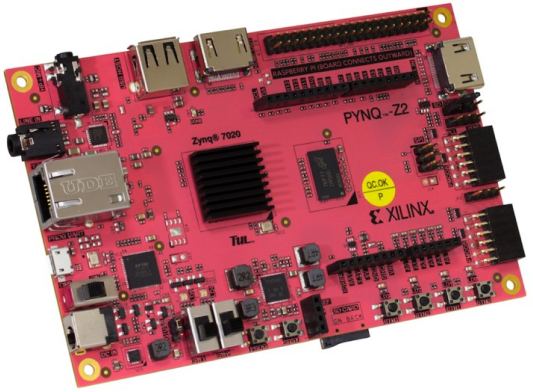
- Spartan 7 (xc7s50csga324-ish):
  - 2.7 Mb of BRAM
  - 120 DSP slices
  - 52K logic cells*
- Dev Board also has 1024 Gb (128 MB) of DRAM
- At commercial price is about 200 USD
- The 1969 Apollo Guidance computer had about:
  - Equivalent of about 1000 logic cells
  - 32 Kb of RAM
  - Cost about 1.5 million 2024 USD a piece.

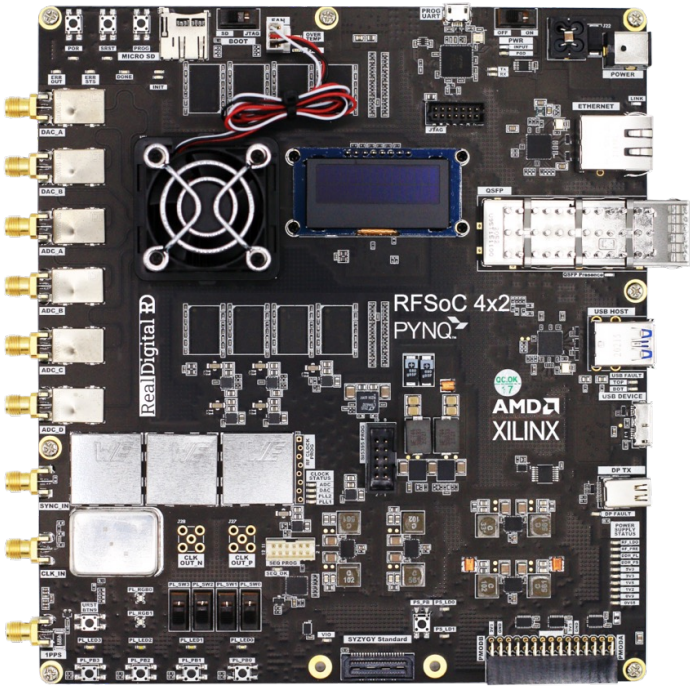
**"logic cell" is a vague term used to compare Xilinx/AMD FPGAs to other vendors. There actually is no such thing as a "logic" cell in Xilinx architecture*

https://docs.amd.com/v/u/en-US/ds180_7Series_Overview

The Appearance of Systems on Chips (SoCs)

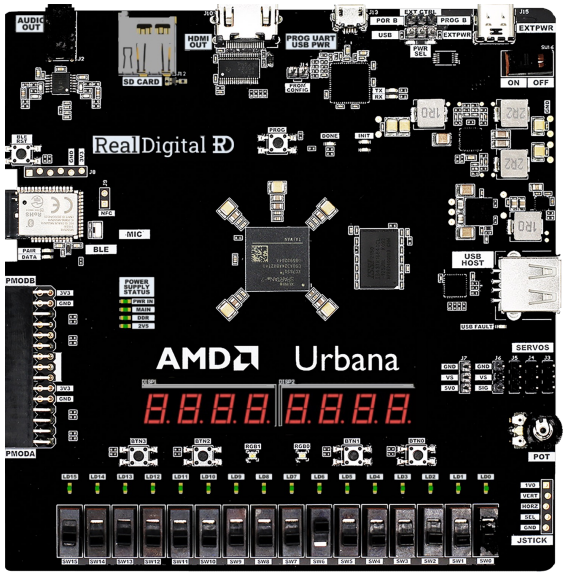
- The last ~10 years have seen the proliferation of Systems on Chips is a whole new epoch
- Many complicated and different types of digital (and now analog) fabrics are all on one single chip





6.205 FPGA

- Spartan 7 (xc7s50csga324-ish):
  - 2.7 Mb of BRAM
  - 120 DSP slices
  - 52K logic cells*
- Dev Board also has 128 MB of DRAM

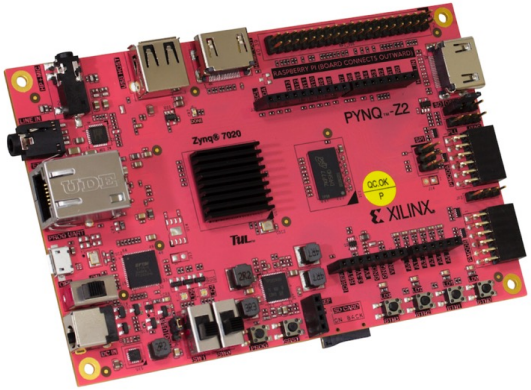


”logic cell” is a vague term used to compare Xilinx/AMD FPGAs to other vendors. There actually is no such thing as a “logic” cell in Xilinx architecture

https://docs.amd.com/v/u/en-US/ds180_7Series_Overview

6.S965 Zynq 7000

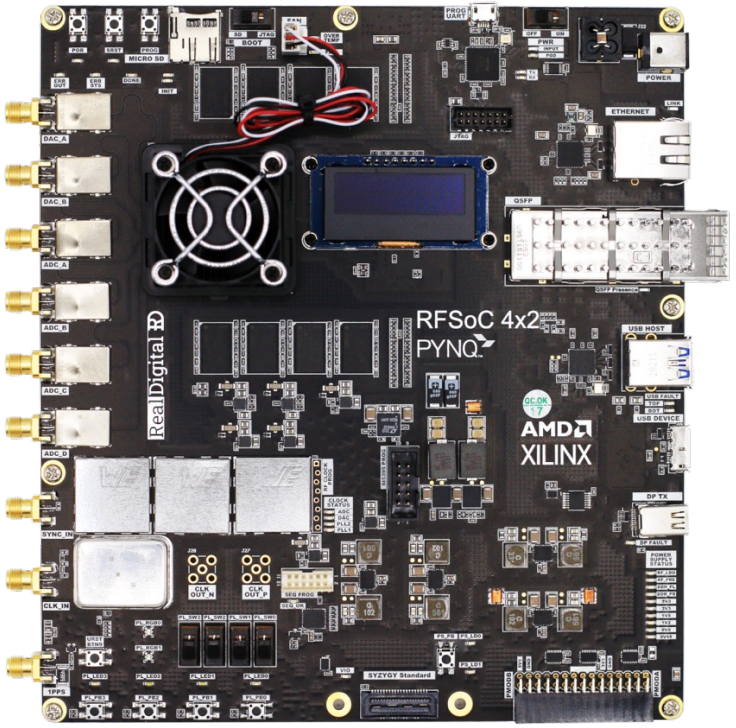
- Series 7000 XC7Z020:
  - 5.04 Mb of BRAM
  - 220 DSP slices
  - 85K logic cells
  - Two 650 MHz A9 ARM processors
  - High-speed interconnects between two resources
- Board has 512 MB of DDR3



https://www.amd.com/en/products/adaptive-socs-and-fpgas/soc/zynq-7000.html

6.S965 RFSoC

- UltraScale+ ZU48DR:
  - 38 Mb of BRAM
  - +22Mb of UltraRAM
  - 4272 DSP slices
  - 930,000 Logic Cells
  - Four 5-Gsps 14 bit ADCs
  - Two 10-Gsps 14 bit DACs
  - Four 1.3 GHz ARM 53 processors
  - Two Real-time 533 MHz ARM processors



- Board has 4GB of DDR4 for FPGA portion (”PL”) and 4 GB of DDR4 for processors (“PS”)

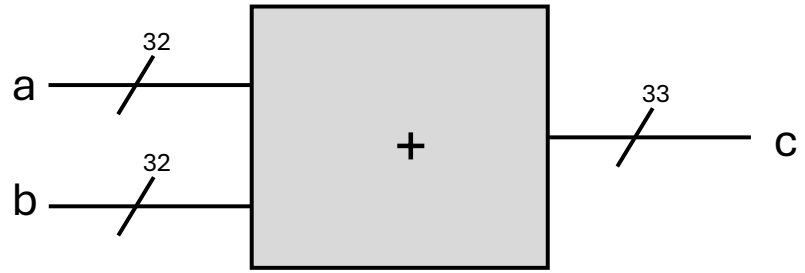
https://www.amd.com/en/products/adaptive-socs-and-fpgas/soc/zynq-ultrascale-plus-rfsoc.html#tabs-b3ecea84f1-item-e96607e53b-tab

Simulation Side of Things

- The capabilities of new digital new systems are so great that the size of the teams working on any one given project has ballooned.
- Validating and/or verifying a digital design used to be something that everyone would do as they designed.
- Now there are whole teams and branches of large companies that exist solely to **verify** designs before they are finalized.

Before spending 100 million dollars…

- How do you verify a design will work?
- Consider a device that adds two 32 bit numbers.



- There are 1.84×10^{19} input possibilities, each with a correct output.
- If you verified 1 billion input/output combinations per second it would take ~600 years to fully verify the design
- And this is just a simple adder…

Conclusion

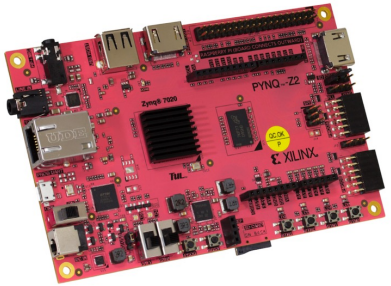
- There's a lot of stuff we can't cover in 6.205 so this is an attempt to do that.
- To be truthful I don't think even one class can do it…
- So what we'll figure out this semester, is what can we do in one more class.
- It'll probably still be a mess, but I'm hoping less so than last year

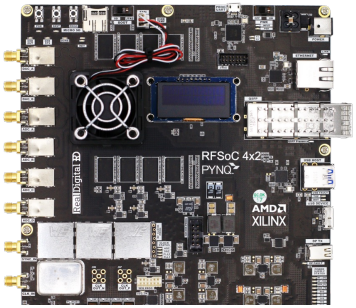
Concepts and Techniques

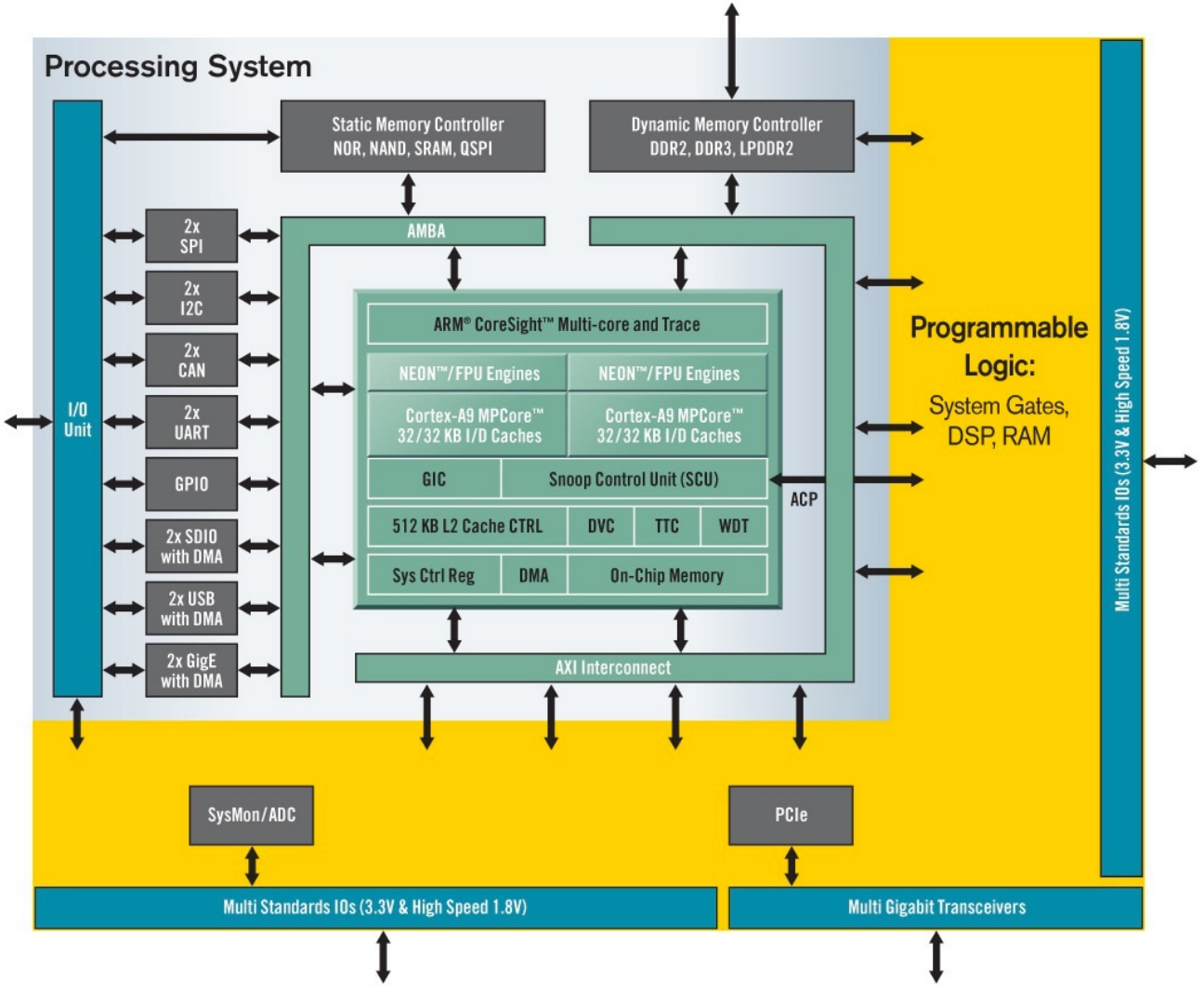
Things we'll be doing...

Zynq

- For SoCs, we'll be using the Zynq platforms by Xilinx/AMD
  - FPGAs
  - Processors
  - other stuff

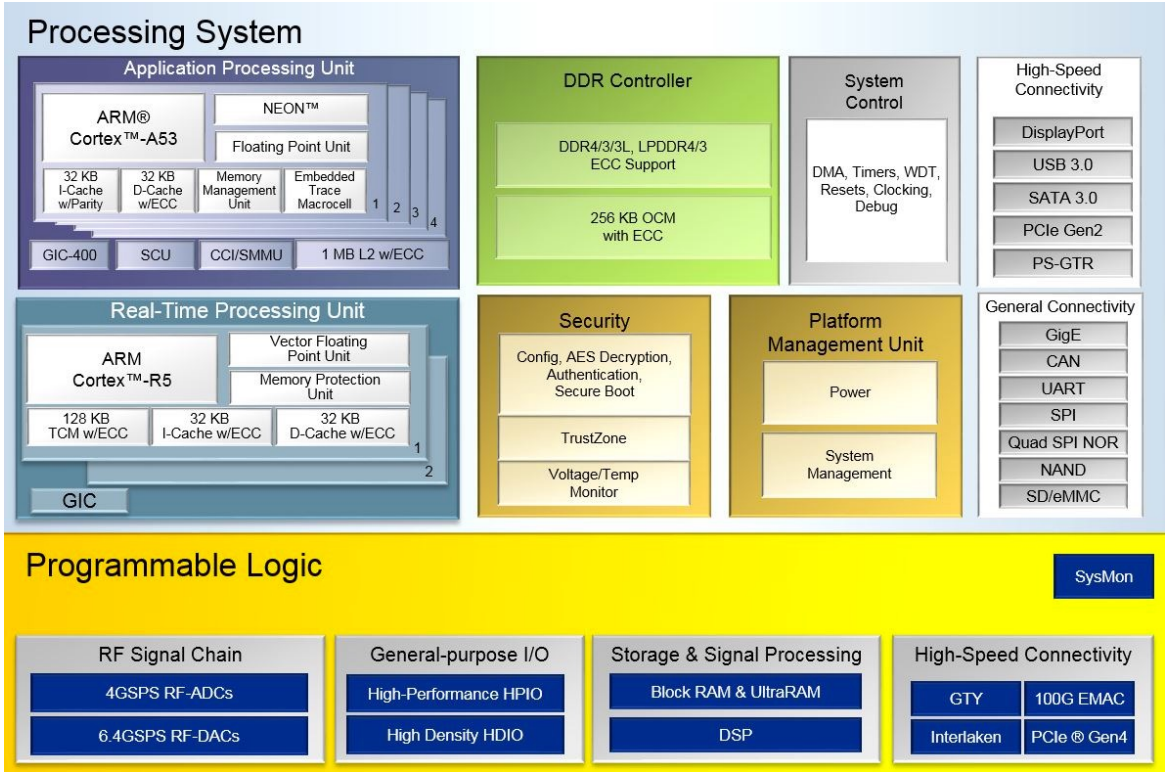






Zynq on RFSoC

- That ”other stuff” can include lots of things:
  - ADCs
  - DACs
  - Hardened MIGs
  - Security
  - Error…
    - Detectors
    - Correctors
  - …

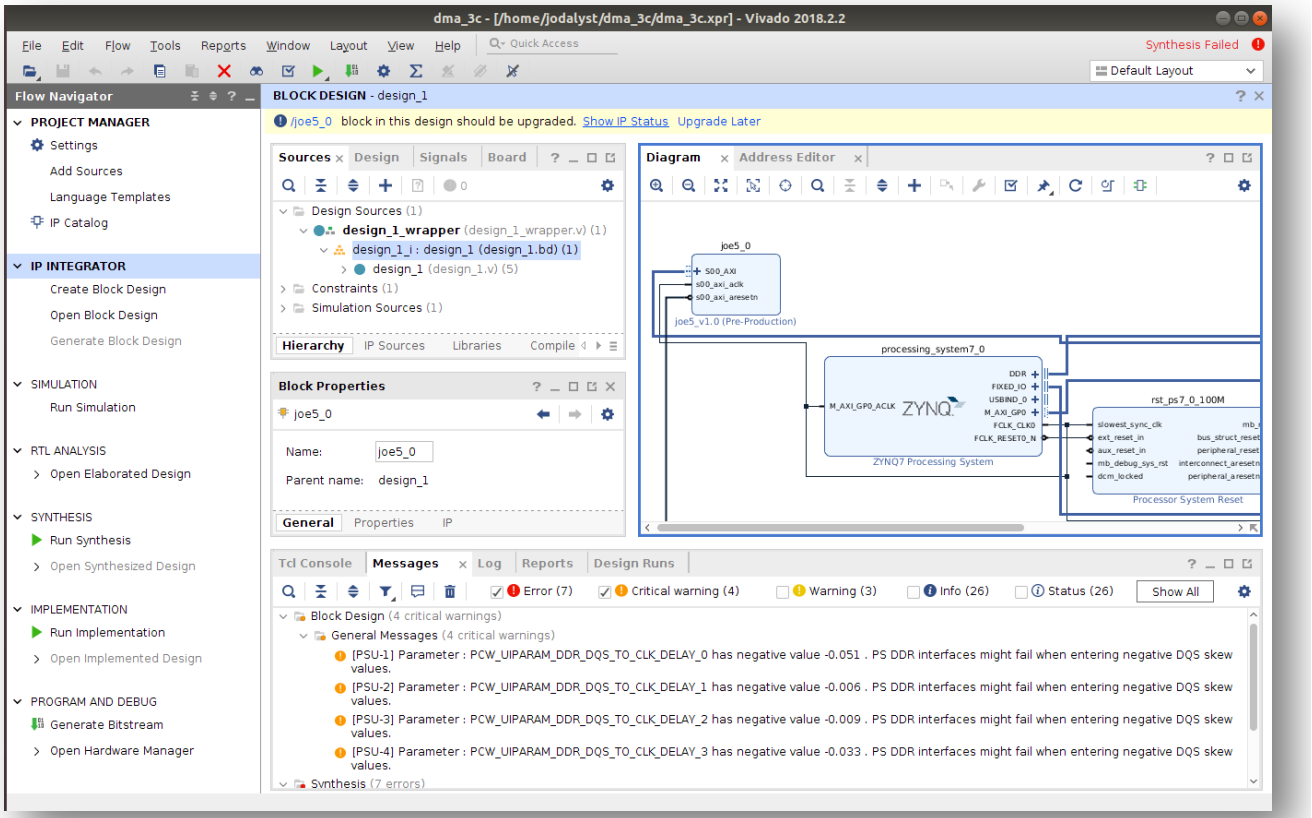


Xilinx/AMD Vivado…

- We’re going to have to work more with Vivado including going into their GUI
- The Zynq frameworks are a bit more vendor-locked because of all the interconnects
- So we need to mess with Vivado a lot more and other frameworks

Vivado Block Design

- Build designs using many tools including a block design editor
- Still also use:
  - Verilog
  - SystemVerilog
  - Waveforms
  - Etc…



Vivado Block Design

- Vivado *really* wants you to use their block design workflow for all Zynq type designs*

- It sounds good at first…

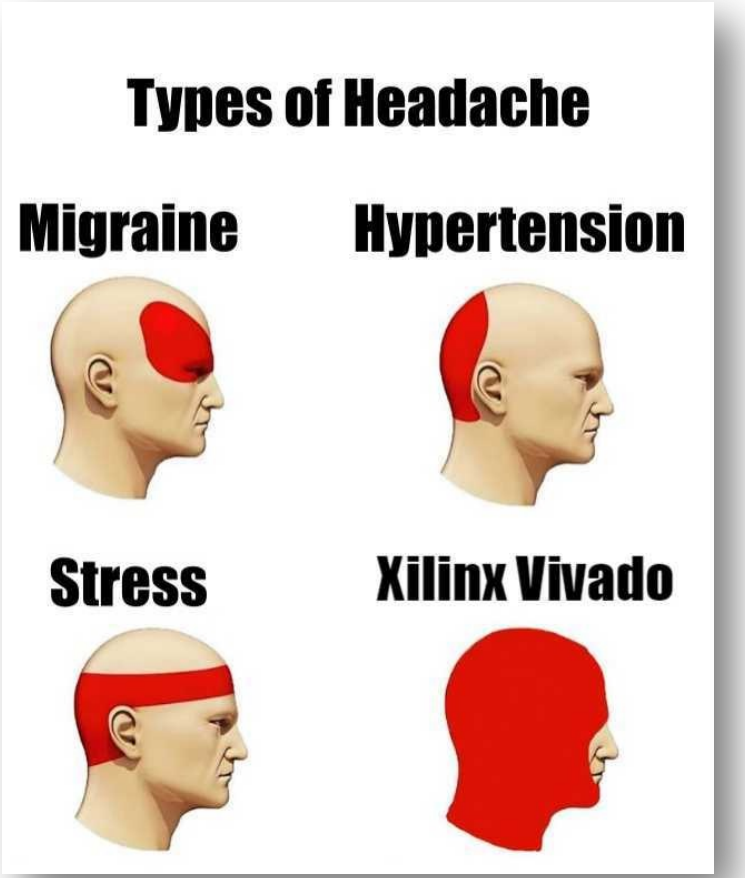
**there are laborious workarounds*

Vivado GUI

```
22
23  module top_level(    input wire clk,
24                       input wire rst,
25                       input wire [3:0] sw,
26                       output logic [3:0] led
27
28      );
29      always_ff @(posedge clk)begin
30          if (rst) be
31      end
```

before
begin

v2025.1



Whereas vscode with some random plugin:

```
22
23  module top_level(    input wire clk,
24                       input wire rst,
25                       input wire [3:0] sw,
26                       output logic [3:0] led
27
28      );
29      always_ff @(posedge clk)begin
30          if (rst) be
31      end
32  endmodule
33
```

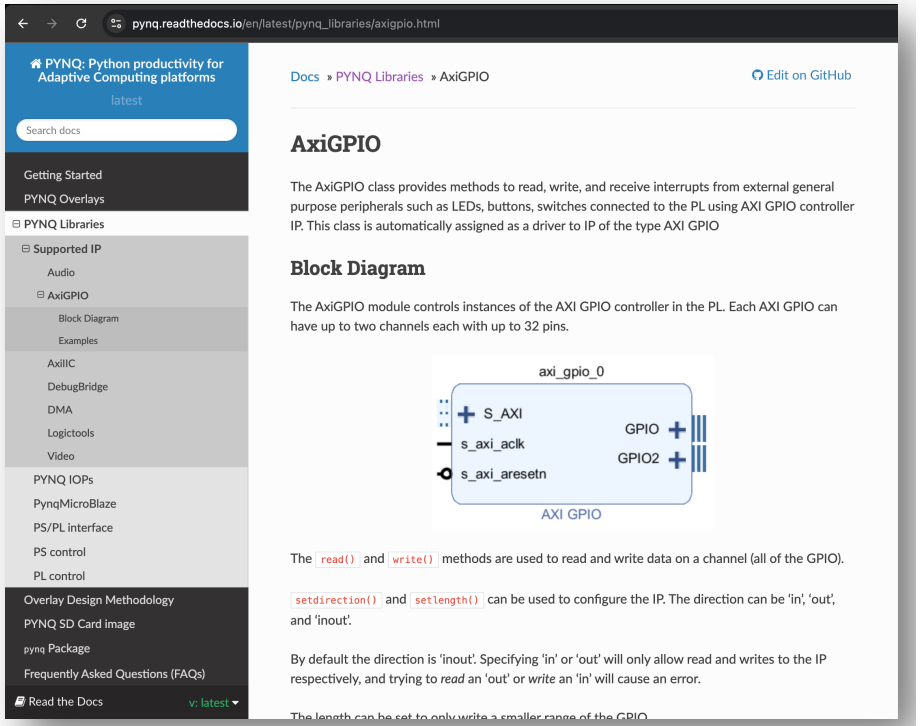
abc begin
begin begin/end
byte byte

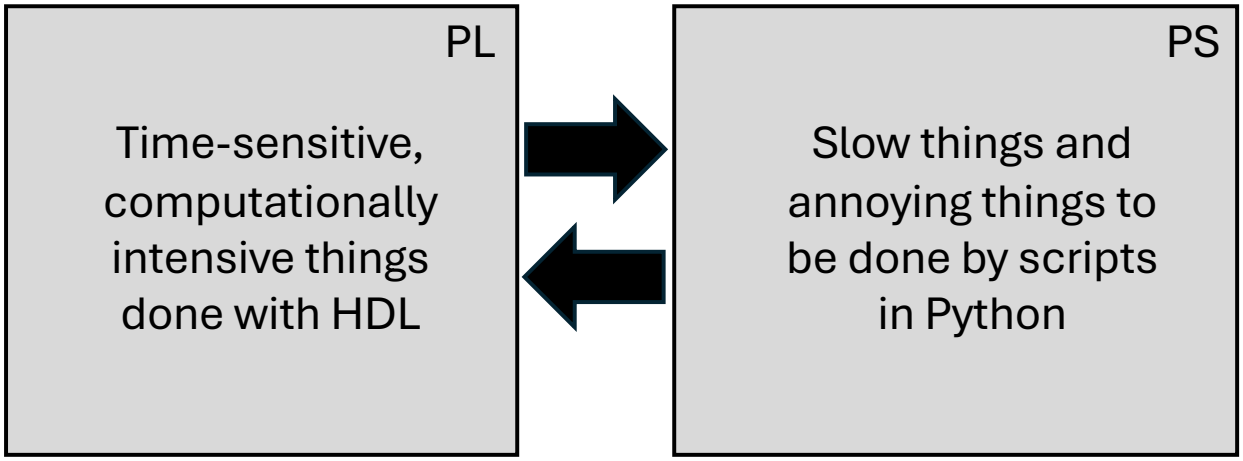
Getting to be a good hardware engineer…

- Means getting to be good, or at least passable, with bad software.
- Almost every commercial tool in the hardware space is frustrating to use (some more than others, of course)

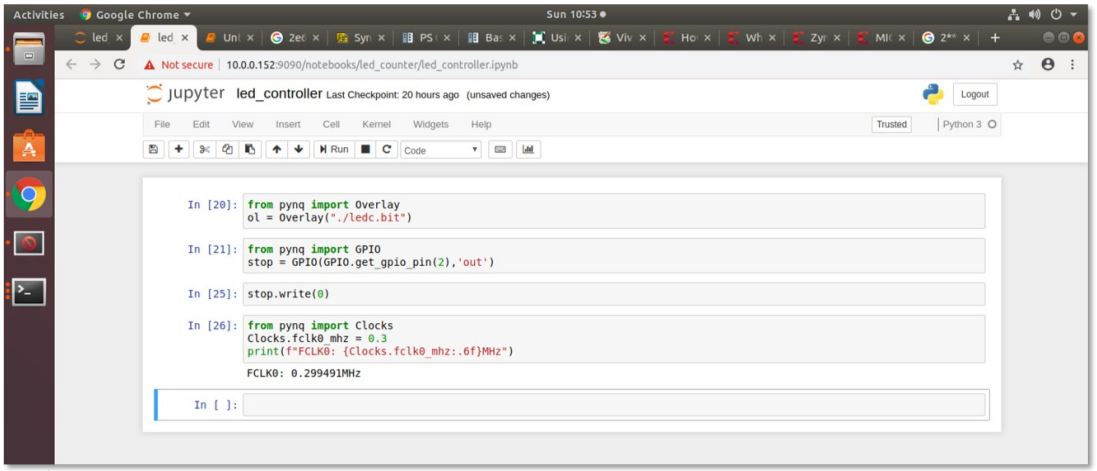
PYNQ: Python-Zynq

- On the software side…
- We'll use the Pynq framework to run Python on the ARM cores of the Zynq



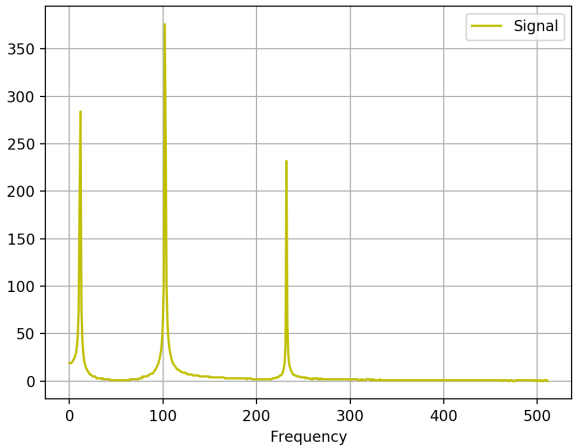


PYNQ: Python-Zynq



- Run Jupyter notebooks
- Decently documented API calls to bridge the software-hardware boundary (which we will explore)
- Actually is stable and has a developer community

Using Python to plot the result of a FFT built in the FPGA
OMG:



AMBA/AXI

- We'll have to go into AMBA and AXI more:
- AMBA: Advanced Microcontroller Bus Architecture
  - AXI is a part of the standard that Xilinx uses in a lot of their modules.

DRAM

- We'll need to go into details of DRAM again. (we're actually bringing that into 6.205 more anyways)

- So we'll spend some time on that and the MIGs on the devices and DMA

Digital Signal Processing

- We need to spend some time on signal processing topics. That is going to require some theory.

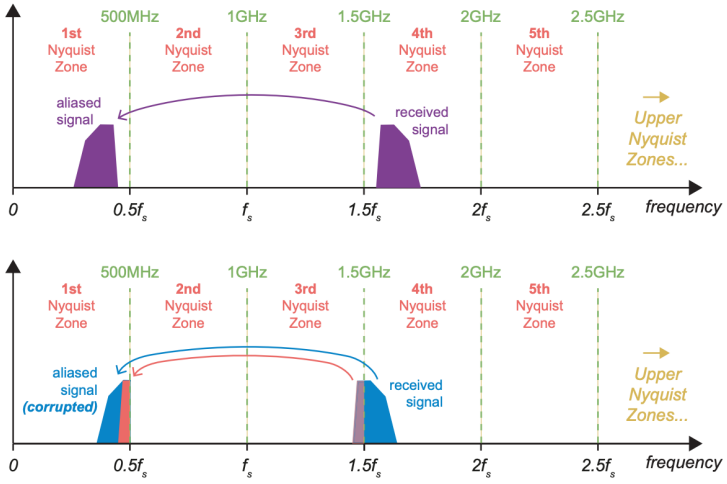


Figure 4.6: Successful and unsuccessful direct downconversion using Nyquist Zones.

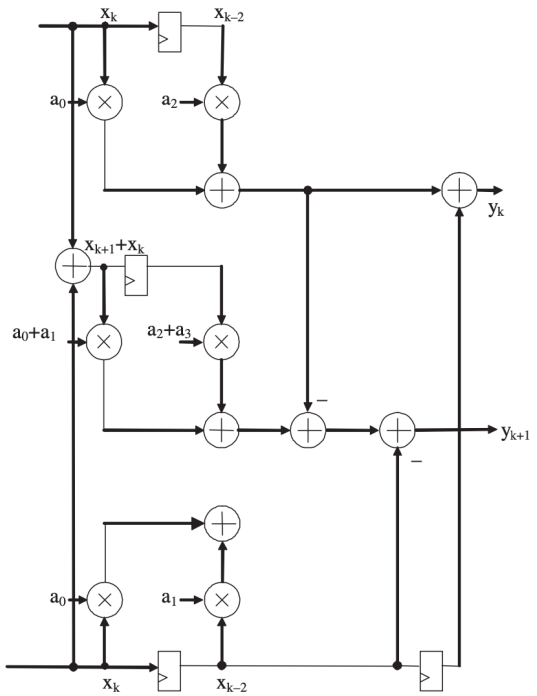


Figure 8.21 Reduced block-based FIR filter

Universal Verification Methodology

- As digital designs have gotten more and more complex the infrastructure around testing them has grown at a similar fashion.
- The codebases for testing and verifying a design are often far larger than the design itself
- And testing/verifying designs is annoying and hard work. Lots of annoying work like:
  - making sure inputs and outputs are controlled properly
  - Monitoring results
  - Testing sufficient space of possibility

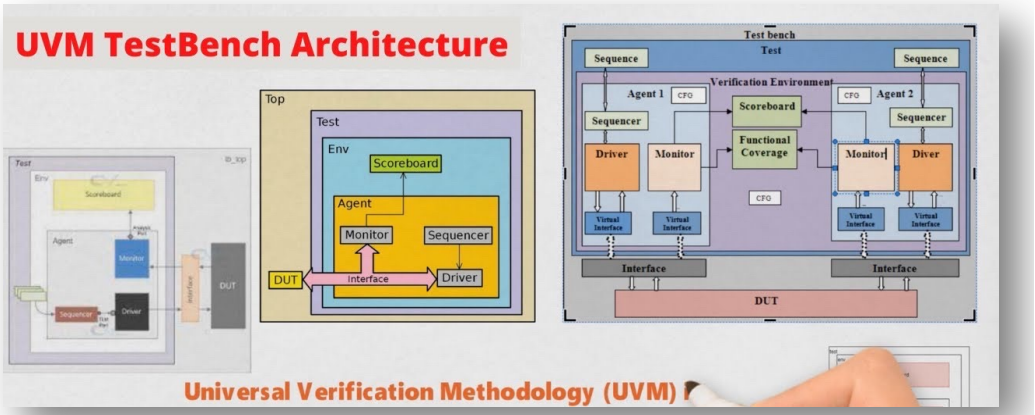
Universal Verification Methodology

- UVM is basically an organizational structure that many modern teams adhere to to ensure all the testing and verification infrastructure is:
  - Scalable
  - Portable
  - Documented
- It is standardized within the SystemVerilog language

UVM

- UVM itself is really a derivation and improvement on prior frameworks including OVM
- UVM is an agreement by all the industry leaders on a common set of testing frameworks
- But the core ideas of UVM are extend beyond the framework. This includes ideas of:
  - Drivers
  - Monitors
  - Scoreboards
  - Coverage
  - Factories
  - Etc…

Teaching UVM



- UVM is a framework with a utility that becomes apparent in large scale projects and designs.
- For small teams it is a harder sell and that's kinda the situation we're in within this class.
- Also it is standardized to SystemVerilog and not the parts of SystemVerilog we use in 6.205, but the really OOPy Java-esque portions of SystemVerilog
- So…here's an alternative idea…

https://www.youtube.com/watch?app=desktop&v=JRfmSv5INP8

Cocotb

- Python library
- Runs right on top of Icarus Verilog of Verilator
- Automates a lot of the annoying portions of writing a testbenches for your .sv files
- Avoids needing to write any simulation code in SystemVerilog at all
- Actually decently documented

Cocotb is actually pretty cool

- It has been around a while, and has stabilized to a point that it is worth teaching.
- It is in Python (which you all should know)
- It links directly into the Python universe (numerical packages useful for models for verification, graphical packages for visualizing data and results)
- At a minimal level it makes getting basic testbenches up and running easier

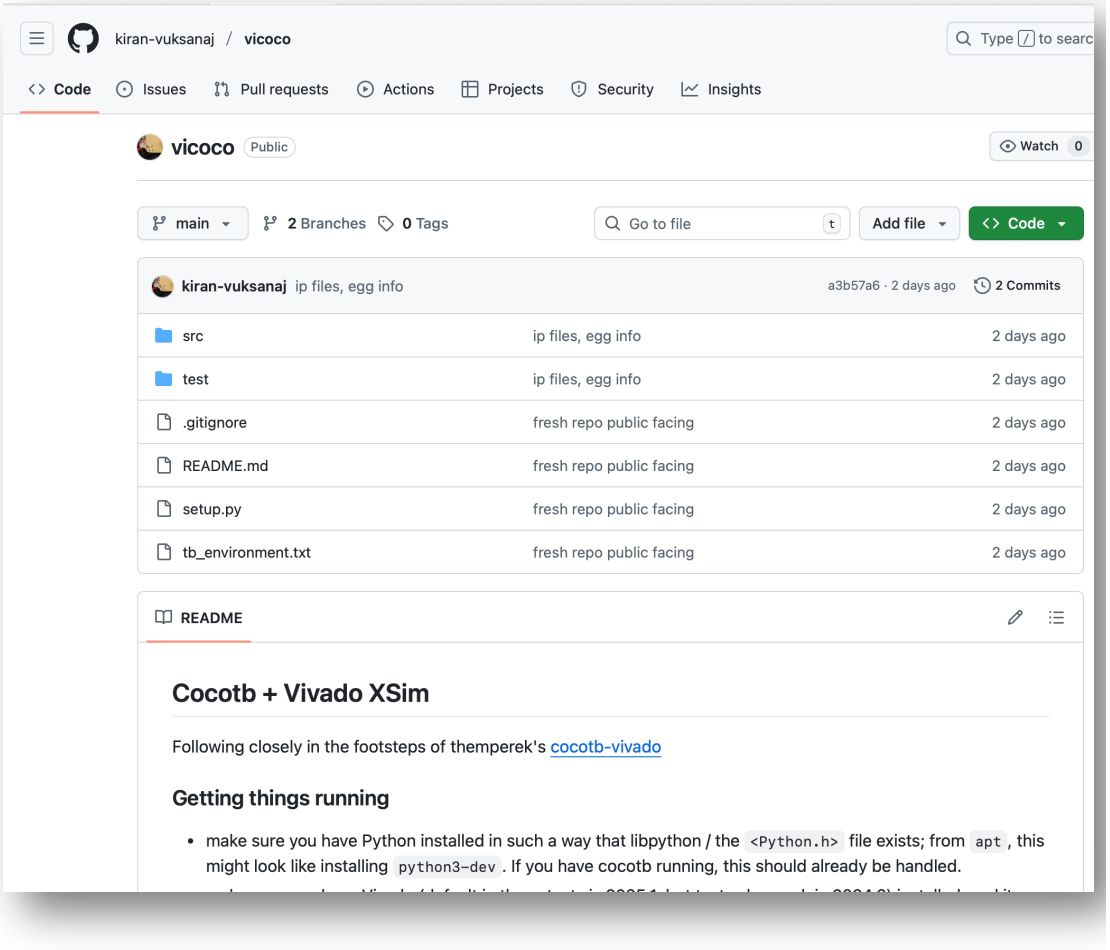
Cocotb is Updating at Some Point

https://docs.cocotb.org/en/development/upgrade-2.0.html

- Unless we decide otherwise, we'll just stay at **cocotb 1.9.2** for this fall since I'm sure 2.0 will break stuff

Vivado + Cocotb = Vicoco

- Kiran Vuksanaj, who was kinda the half-TA for this class last year and is TAing 6.205 this fall wrote hooks between Vivado and Cocotb so *we should* be able to actually test encrypted Xilinx IP this year with cocotb



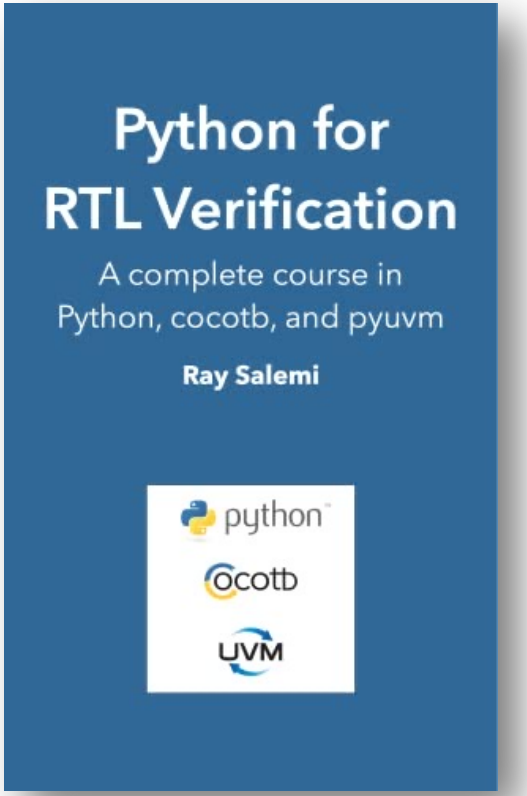
https://github.com/kiran-vuksanaj/vicoco

Cocotb + Other Things

- Using other libraries and code we write, I'd like to build up many of the concepts of a modern verification framework
- Along the way, we'll explore what those are in UVM and what they mean
- Doing this will also let us avoid having to go heavy into SystemVerilog

This is *not* my idea either

- Others have tried and/or are trying to bring UVM frameworks into Cocotb
- They're not there yet and the docs/tutorials they do have are all largely concerned with listing off the various types of sub, sub-sub, and sub-sub-subclasses rather than actual uses

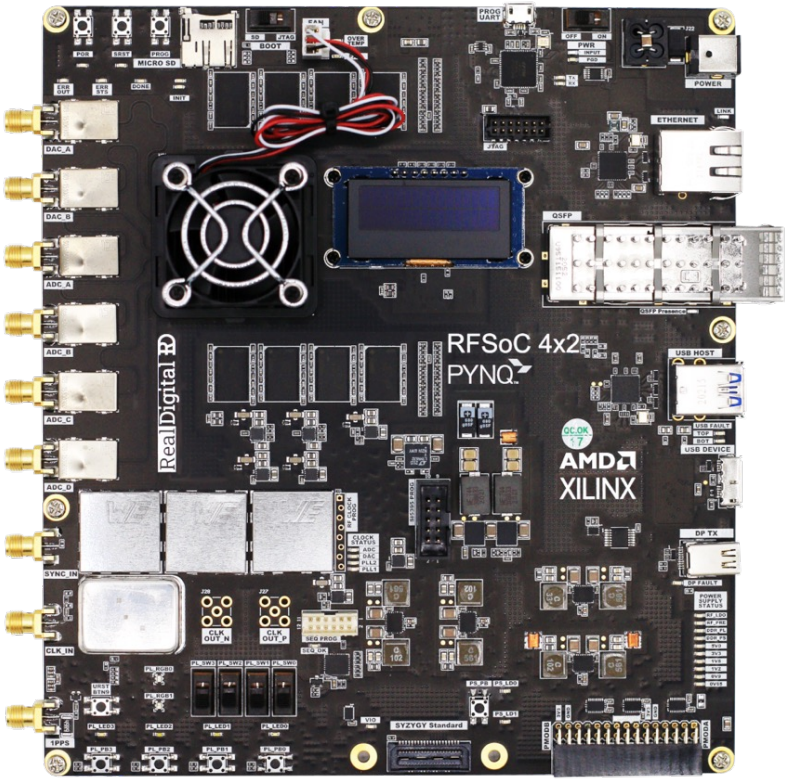


Final Projects

- I'll need a final report and video from you and documented code, but we're not going to have CI-M level presentations and pre-reports and things.
- You can work in teams, but if absolutely necessary you can work on your own, if absolutely necessary.
- We'll have a somewhat limited budget (few thousand freedom bucks for class in total)

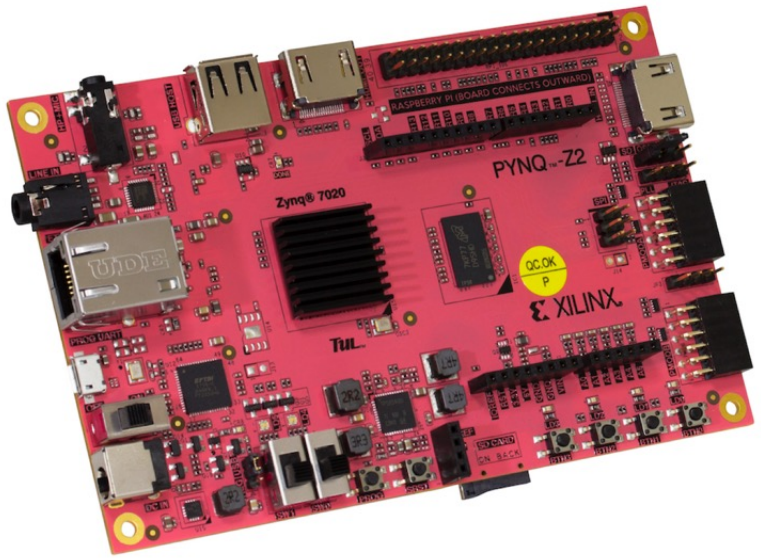
RFSoC

- Lots of resources if you want to do anything with RF, this would be the first choice board
- Been around for a lot less so documentation isn't as mature



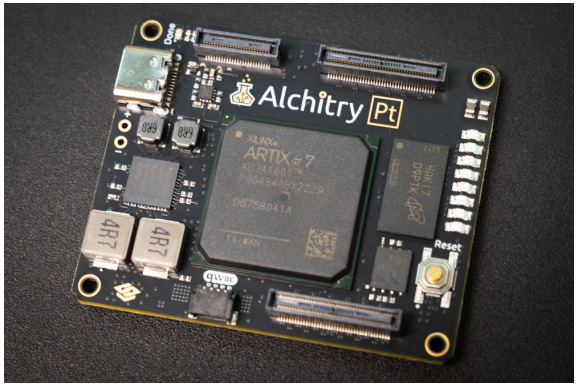
Pynq Z2

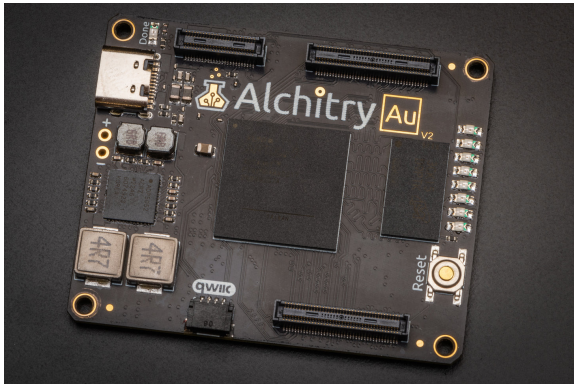
- Likely good for video
- Video In and Video Out Ports
- Use up through Week 3 in labs (that's the plan right now)



Alchitry FPGA Boards

- Interesting because of price point
- Very small.
- Also have decent, tested non-Xilinx MIG for their DDR3
- Also have many, many properly routed and matched output traces for external connections…so if anybody wants to do an interface with them for a project/is a possibility





Ideas

- Deep-TEMPEST: Using Deep Learning to Eavesdrop on HDMI from its Unintended Electromagnetic Emanations

So What's Next?

- First Week's assignments out on Friday ~12pm
- Few parts:
  - Learning/remembering Cocotb
  - Do some simple verification on a module
  - Integrate that module into the cheaper Pynq Zynq-7000 board to get practice with the whole software/hardware integration thing.
- Next week we'll look at Cocotb, simulations, etc.. in more depth as well as the Zynq system