6.S965 Digital Systems Laboratory II

Lecture 1: Introduction

Course Overview

- Instructor: Joe Steinmeyer
- Quasi-TA: Kiran Vuksanaj
- Prerequisites: 6.205
- Units: 2-8-2
- Lecture 3:30-4:30(ish) MW in 5-234
- Lab is rolling (similar to 6.205).
- Site: https://rfsoc.mit.edu/6S965/F24
- **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 Content

- Keep Studying FPGAs and associated technologies including:
 - Systems on Chips (SoCs, RFSoCs)
 - Common Peripherals (DRAM, transceivers, power supplies, etc...)
- Study some signal processing concepts
- Investigate and study modern verification techniques:
 - Universal Verification Methodology (UVM)
 - Cocotb
- Books: We'll use several:
 - <u>Software Defined Radio with Zyng Ultrascale+ RFSoC</u> (linked on site)
 - Few others as needed.

Schedule

- Sept 3 to Oct 25: Weekly assignments and lectures
 - Lectures Monday/Wednesday 3:30-4:30
 - Weekly assignments: Friday-to-Friday release-due cycle
- Oct 25 to Dec 11: 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 6 and due on Sept 13)
- 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.

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 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

Couple Other Things

- We are co-using the lab space with 6.205.
- You will very likely need to use the lab in-person more than you did in 6.205. The Zynq boards with the Pynq framework are best worked with over a local network which we'll have set up.
- I'm not letting you take the RFSoC boards outside the lab.

Motivation for Course

- In 1969, MIT had approximately one Digital Electronics Class (and occasionally a special subject in advanced topics)
- The field has expanded immensely. Areas of work which were once a single unit in a class are now full career/research areas.

		FALL IDIG	
		G. HOM FALL, 1969	
		and an IN- and a second and and and	F
Finfor		and the second sec	
6.711T	Digital Systems	Project Laboratory	
	Prereq.: 6.01, 6.70 Year: U (1, 2)	3	7-2
Introduc	tion to some of the	e basic techniques and dev	
used in	the design and im	plementation of digital co	om-
puting a	and control system	ns. Preliminary work by	va
combina	tion of classes,	homework, and associa	ted
Detailed	study of logic des	th digital integrated circu sign, including Boolean al	lits.
bra, syn	thesis of combination	ational logic iterative lo	oric
realizatio	ons, and techniques	s for description and synthe	
of seque	ential circuits. oth	per topics such as man	
with mi	crocircuit logic el	mplementation of all proje lements. Oral presentat	ects
and writ	ten report on com	pletion of project. (Registra	ion
limited to	40 students.)		oxel
initica to	40 staachts.)	Ire	oxel

2024 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
- 6.206: Microcontrollers
- 6.601: Analys and Design of Digital Integrated Circuits
- Etc...

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



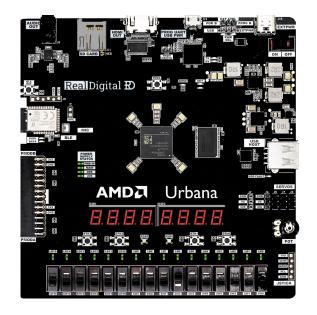
Lab kit 1990 aka "digital death"



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.

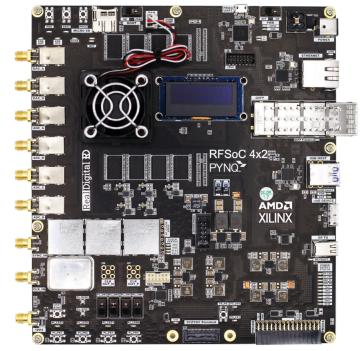
*"logic cell" is a vague term used to compare Xilinx/AMD FPGAs to other vendors. There actually is no such thing as a "logic" cell in Xilinx architecture

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

The Appearance of Systems on Chips (SoCs)

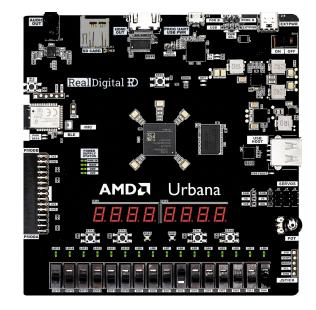
- The last ~10 years have seen the proliferation of Systems on Chips is a whole new epoch
- Many complicated and different types of digital (and now analog) fabrics are all on <u>one single chip</u>





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

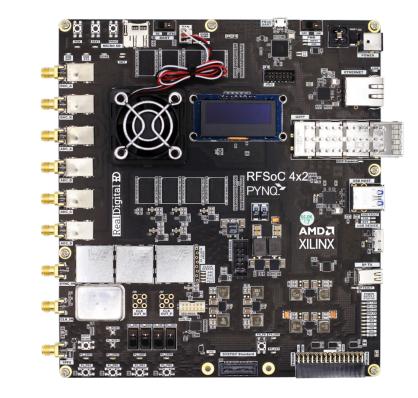
- Series 7000 XC7Z020:
 - 5.04 Mb of BRAM
 - 220 DSP slices
 - 85K logic cells
 - Two 650 MHz A9 ARM processors
 - High-speed interconnects between two resources
- Board has 512 MB of DDR3



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")

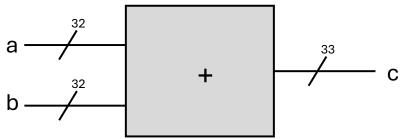
https://www.amd.com/en/products/adaptive-socs-and-fpgas/soc/zynq-ultrascale-plus-rfsoc.html#tabs-b3ecea84f1-item-e96607e53b-tab

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.

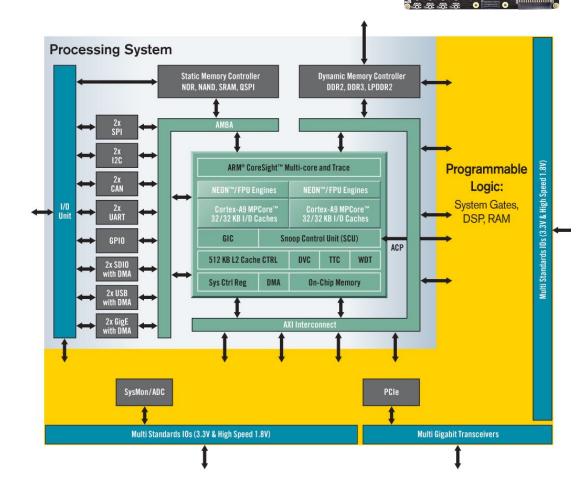
Concepts and Techniques

Things we'll be doing...

Zynq

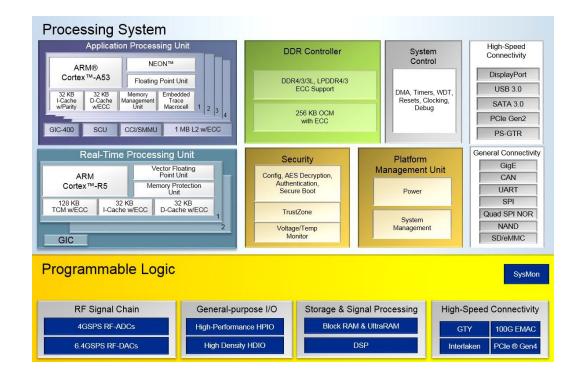


- For SoCs, we'll be using the Zynq platforms by Xilinx/AMD
 - FPGAs
 - Processors
 - other stuff



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 vendorlocked 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...

	ana_se [Inome/Jouddyse/ana	_3C/dilla_3C.xp1] - V1Vado 2018.2.2	
<u>File Edit Flow Tools Reports</u>	Window Layout View Help Q- Quick Access		Synthesis Failed 🏾 🌖
	b 🗹 ▶, 👫 🌣 ∑ 🖄 🖉 🔏		📰 Default Layout 🛛 🗸 🗸
Flow Navigator 🗧 🗧 🗧 🚆	BLOCK DESIGN - design_1		? ×
PROJECT MANAGER	Ijoe5_0 block in this design should be upgraded. Show IP !	Status Upgrade Later	
🔅 Settings	Sources × Design Signals Board ? _ []	Diagram × Address Editor ×	200
Add Sources			
Language Templates			의 1D: 🗘
👎 IP Catalog	<pre>>> Design Sources (1) >>> design_1_wrapper (design_1_wrapper.v) (1)</pre>		
✓ IP INTEGRATOR	<pre></pre>	joe5_0	
Create Block Design	> 🗁 Constraints (1)	s00_axi_acik	
Open Block Design	> 🗁 Simulation Sources (1)	ice5 v1.0 (Pre-Production)	
Generate Block Design	Hierarchy IP Sources Libraries Compile 4 ▶ ≡	processing_system7_0	
 SIMULATION 	Block Properties ? _ D 🗅 X	DDR +	
Run Simulation	≠ joe5_0	M_AX[GP0,ACLK ZYNQ USBAD,0 +	rst_ps7_0_100M slowest_sync_clk mb_t
 RTL ANALYSIS 	Name: joe5 0	FCLK_RESETO_N	 ext_reset_in bus_struct_reset aux_reset_in peripheral_reset
> Open Elaborated Design	Parent name: design 1	ZYNQ7 Processing System	 mb_debug_sys_rst interconnect_aresetn dcm_locked peripheral_aresetn
	rarenz name. design_1		Processor System Reset
 SYNTHESIS 	General Properties IP		<u>ک ج</u>
Run Synthesis			
> Open Synthesized Design	Tcl Console Messages x Log Reports Design	n Runs	? _ 🗆 🖾
	🔍 🔍 🗮 🜲 🟹 🖽 🛅 🛛 🗹 🖯 Error (7) 📿 🕒	Critical warning (4) 🛛 🕒 Warning (3) 📄 🚯 Info (26) 🔤 🚯 Status	s (26) Show All 🂠
IMPLEMENTATION	Block Design (4 critical warnings)		^
Run Implementation	 General Messages (4 critical warnings) IBSU 11 Parameter - RCW URARAM DDR DOS TO 	CLK DELAY 0 has negative value -0.051 . PS DDR interfaces might fail when en	toring pogetive DOS show
> Open Implemented Design	values.	CER_DEDAT_0 has negative value 40.001 . PS DDR internates might hair when en	itering negative DQ5 skew
PROGRAM AND DEBUG	 (PSU-2) Parameter : PCW_UIPARAM_DDR_DQS_TO_ values. 	_CLK_DELAY_1 has negative value -0.006 . PS DDR interfaces might fail when en	tering negative DQS skew
III Generate Bitstream		_CLK_DELAY_2 has negative value -0.009 . PS DDR interfaces might fail when en	tering negative DQS skew
> Open Hardware Manager		_CLK_DELAY_3 has negative value -0.033 . PS DDR interfaces might fail when en	tering negative DQS skew \sim

Vivado Block Design

- Vivado really wants you to use their block design workflow for all Zynq type designs*
- It sounds good at first...



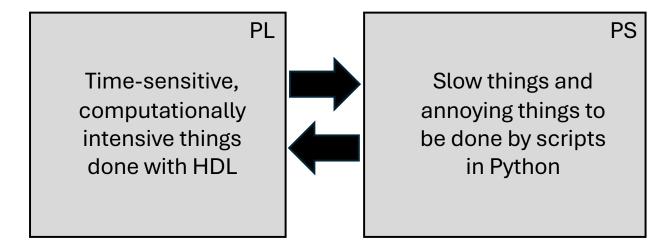
*there are workarounds but it takes a lot of effort and built-up code base

6.S965 Fall 2024

PYNQ: Python-Zynq

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

$m \epsilon o {f C}$ ${f a}$ pynq.readthedocs.io/en/la	atest/pyng_libraries/axigpio.html
PYNQ: Python productivity for Adaptive Computing platforms latest	Docs » PYNQ Libraries » AxiGPIO
Search docs	AxigPIO
Getting Started PYNQ Overlays	The AxiGPIO class provides methods to read, write, and receive interrupts from external general purpose peripherals such as LEDs, buttons, switches connected to the PL using AXI GPIO controller
PYNQ Libraries	IP. This class is automatically assigned as a driver to IP of the type AXI GPIO
Supported IP Audio	Block Diagram
	-
Block Diagram Examples	The AxiGPIO module controls instances of the AXI GPIO controller in the PL. Each AXI GPIO can have up to two channels each with up to 32 pins.
AxillC DebugBridge DMA Logictools Video PYNQ (JOPs PynqMicroBlaze	axi_gpio_0 + S_AXI s_axi_aclk GPIO + III • S_axi_aresetn AXI GPIO
PS/PL interface PS control	The read() and write() methods are used to read and write data on a channel (all of the GPIO).
PL control	
Overlay Design Methodology PYNQ SD Card image pyng Package	<pre>setdirection() and setlength() can be used to configure the IP. The direction can be 'in, 'out', and 'inout'. By default the direction is 'inout'. Specifying 'in' or 'out' will only allow read and writes to the IP</pre>
Frequently Asked Questions (FAQs)	respectively, and trying to read an 'out' or write an 'in' will cause an error.
B Read the Docs v: latest ▼	The length can be set to only write a smaller range of the GPIO

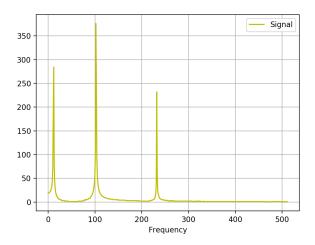


PYNQ: Python-Zynq

ties 🚺	🦻 Google (Chrome 🔻					Sun 10:5	3•						- A •	こ (1)
C	led ×	🧧 led 🗙 🧧 Un	t × 🌀 2e6 ×	🔛 Syn 🗙 📔	B PS(×	🖪 Bas 🗙	📜 Usi ×	😴 Viv 🗙	🗧 Hoʻ X	💈 Wh 🗙	🗧 Zyr 🗙 🔤	🗧 міс 🗙	G 2 ^{*+} ×	+	•••
÷	\rightarrow G	A Not secure 10	.0.0.152:9090/not	ebooks/led_cou	nter/led_cor	ntroller.ipynb								☆	Θ:
		💭 jupyter I	led_controller	Last Checkpoint	20 hours age	o (unsaved ch	anges)						el Logo	ut	
		File Edit V	iew Insert	Cell Kernel	Widgets	Help						Trusted	Python 3	0	
		B + × 4	16 + + 1	Run 📕 C	Code	· •	[ata]								
]		In [21]: In [25]:	<pre>stop.write(6 from pynq im Clocks.fclk@ print(f*FCLK FCLK0: 0.299</pre>	<pre>("./ledc.bit mport GPIO GPIO.get_gpi)) mport Clocks 0 mhz = 0.3 0; {Clocks.f</pre>	") o_pin(2),										
		TU []:													

- Run Jupyter notebooks
- Decently documented API calls to bridge the softwarehardware 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:



6.S965 Fall 2024

AMBA/AXI

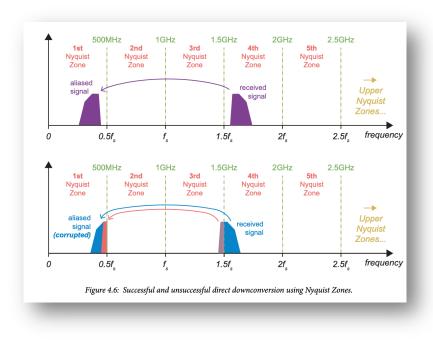
- We'll have to go into AMBA and AXI more:
- AMBA: Advanced Microcontroller Bus Architecture
 - AXI is a part of the standard that Xilinx uses in a lot of their modules.
- I'd also like to look into Wishbone and a few other bus protocols out there

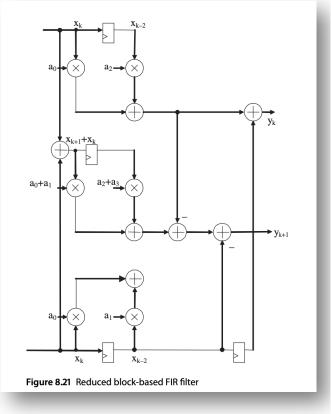
DRAM

- 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.





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 "unsexy" work like:
 - making sure inputs and outputs are controlled properly
 - Monitoring results
 - Testing sufficient space of possibility

Universal Verification Methodology

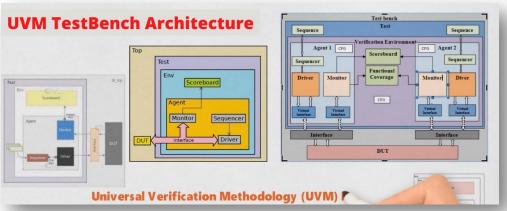


- UVM is basically an organizational structure that many modern teams adhere to to ensure all the testing and verification infrastructure is:
 - Scalable
 - Portable
 - Documented
- It is standardized within the SystemVerilog language

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
 - 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



Use cocotb to test and verify chip designs in Python. Productive, and with a smile.

cocotb is an open source coroutine-based cosimulation testbench environment for verifying VHDL and SystemVerilog RTL using Python.

- 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 + 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, subclasses rather than actual uses

Python for RTL Verification

A complete course in Python, cocotb, and pyuvm

Ray Salemi



So What's Next?

- First Week's assignments out on Friday ~12pm
- Two big portions:
 - Getting started with Cocotb and some testbenching with that framework
 - Building a simple hardware/software system on the Pynq Zynq-7000 board
- Next week we'll look at Cocotb in more depth as well as the Zynq system