# Unit 0
# Course Introduction

EL-GY6143: INTRODUCTION TO HARDWARE DESIGN

PROF. SUNDEEP RANGAN, SIDDHARTH GARG

# Outline

- Course Admin
- What is Custom Hardware and What Can it Do?

# Course Details

- Instructors
  - Sundeep Rangan, 370 Jay St, 9-th floor
  - Siddharth Garg, 370 Jay St, 10-th floor
- TA's:  TBD
- Office Hours:
  - Instructor:  Either Siddharth or Sundeep.  Wednesdays, 3-4
- Time and Location:
  - Jacob's Hall, 6 Metrotech Room 774
  - Tuesdays 2-4:30

# Learning Objectives

- ❑ Identify tasks for hardware acceleration
- ❑ Design, simulate, and evaluate designs for hardware accelerators
  - Called Intellectual Property or IPs
  - Use state-of-the-art tools
  - High-level synthesis (HLS) and AI
- ❑ Integrate IP into processor-based systems for real-world applications
- ❑ Deploy and test systems on FPGA boards

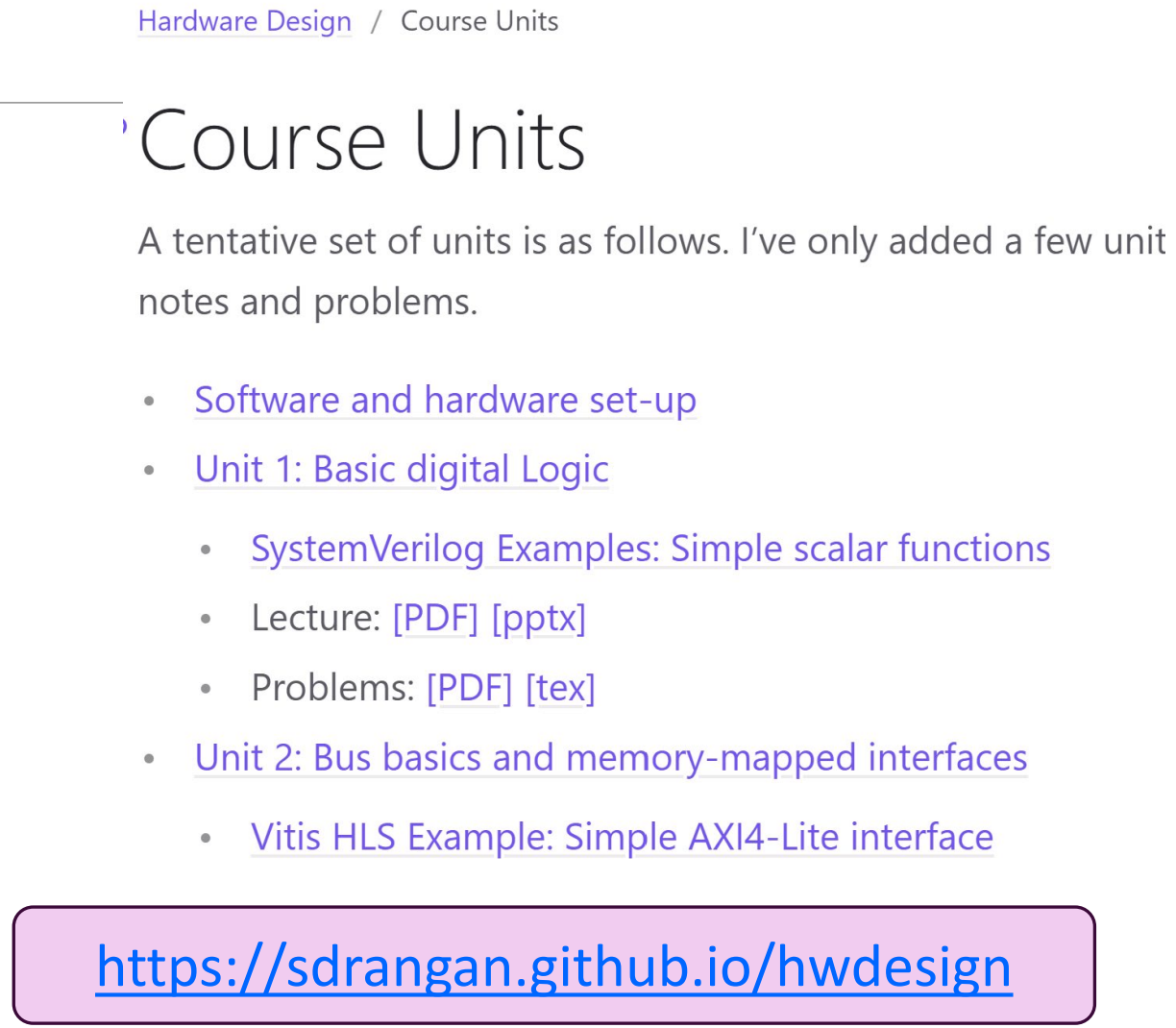
# Pre-Requisites

*Anyone can learn hardware*

- No prior hardware experience is required!
- No hard pre-requisites!
- Just have
  - Some software experience (e.g., C, python)
  - Desire to learn
- Many possible disciplines
  - Robotics, wireless, scientific computation, cryptography, networking, …

# Course Webpage

- ❑ All the material for the class is on GitHub
- ❑ Demos:
  - SystemVerilog
  - Vitis HLS
  - Python
- ❑ Lecture notes
- ❑ Problems, Lab assignnments
- ❑ Repository is work in progress
  - Re-sync periodically



Hardware Design / Course Units

## Course Units

A tentative set of units is as follows. I've only added a few unit notes and problems.

- Software and hardware set-up
- Unit 1: Basic digital Logic
  - SystemVerilog Examples: Simple scalar functions
  - Lecture: [PDF] [pptx]
  - Problems: [PDF] [tex]
- Unit 2: Bus basics and memory-mapped interfaces
  - Vitis HLS Example: Simple AXI4-Lite interface

https://sdrangan.github.io/hwdesign

# Software Set-Up

- To perform this course, you will need to:
- Install Vitis and Vivado
  - Software tools for designing hardware
- Clone the class repository
- Optional:  Purchase an actual FPGA board
  - See next slide
- Vitis and Vivado options:
  - Run locally on a Windows machine
  - MacOS:  You must run it in Linux virtual machine
  - Use NYU server.  Details to be provided shortly

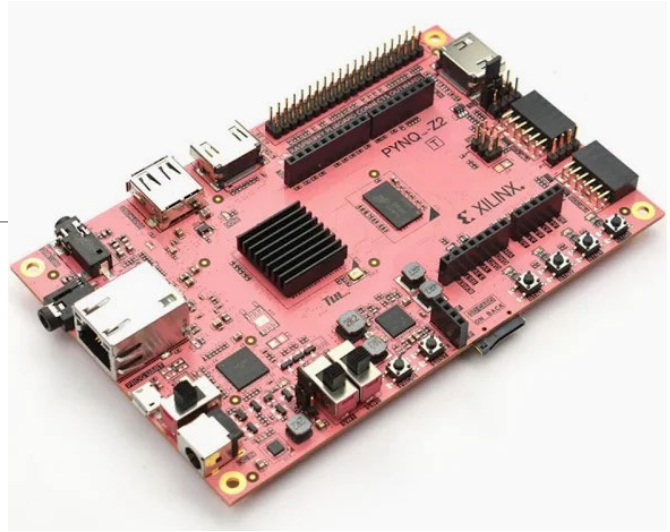
Hardware Design / Course Units / Getting Started

## Getting Started

Before stating the units, you will need to:

- Install Vitis and Vivado: These are the m
  synthesize and simulate the hardware a
- Clone the class GitHub repository: The
- Purhcase and set-up and FPGA board: I
  you will have to purchase a board and

# FPGA Boards




- Projects and labs can be deployed on actual hardware
  - Currently optional due to price
- We suggest:  Pynq-Z2 board
  - $300
  - USB connection to your laption
  - Pynq:  Simple python interface
  - Low-end, but ideal for teaching
- Other boards may be available in lab
  - Ex: RFSoC 4x2 for wireless communications

```
from pynq import Overlay
overlay = Overlay("../scalar_fun_pynq
```

```
# Input parameters
a = np.random.randint(low=0, high=128)
b = np.random.randint(low=0, high=128)

# Sets the two inputs
ip.register_map.a = a
ip.register_map.b = b

# Start the execution
ip.register_map.CTRL.AP_START = 1
```

# Tentative Grading

- Points
  - Homework + Labs: 25%
  - Midterm:  25%
  - Final:  25%
  - Project:  25%
- Homework will be mostly written problems
- Labs will be mostly VitisHLS
  - Approximately 5 Vitis HLS assignments
  - Approximately 5 written assignments
- Midterm and Final are in class and

# Using AI

- It is encouraged!
  - Use it for labs, problems, projects
- But learn to use it well:
  - It still makes mistakes.
  - Learn to test the results
  - Write clear, specific prompts
  - Use it as a tutor
  - Have it generate questions
  - Explain results
- On the midterm & final there are no aids!

Can you write SystemVerilog code to implement a linear function + ReLU:

y = max(0, a*x+b)

Sure! Here's a simple SystemVerilog module that computes $y = \max(0, a \cdot x + b)$ using combinational logic:

Systemverilog

```
module linear_relu #(
    parameter WIDTH = 16
)(
    input  logic signed [WIDTH-1:0] x,
    input  logic signed [WIDTH-1:0] a,
    input  logic signed [WIDTH-1:0] b,
    output logic signed [WIDTH-1:0] y
);

    logic signed [2*WIDTH-1:0] mult;
    logic signed [2*WIDTH-1:0] sum;

    always_comb begin
        mult = a * x;
        sum  = mult + b;
        y    = (sum > 0) ? sum[WIDTH-1:0] : '0;
    end

endmodule
```

# Tentative Syllabus 1

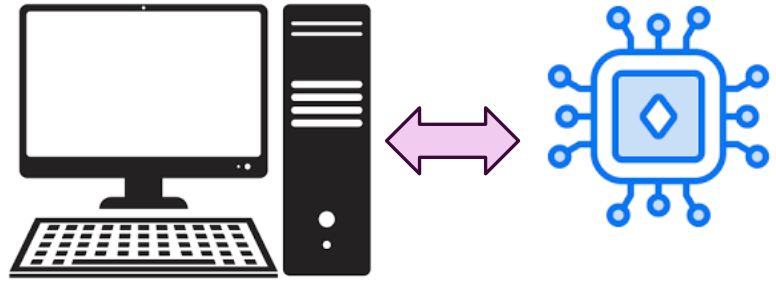
| Week | Date | Material |
|---|---|---|
| 1 | 1/20/2026 | Course Admin<br>Unit 1. Basic digital logic |
| 2 | 1/27/2026 | Unit 2. Bus basics and processor interface |
| 3 | 2/3/2026 | Unit 3. FIFO interfaces |
| 4 | 2/10/2026 | Unit 4. Loop optimization |
| 5 | 2/17/2026 | Unit 5. Arithmetic representation and hardware units |
| 6 | 2/24/2026 | Unit 6. Systolic Arrays |
| 7 | 3/3/2026 | Buffer |
| 8 | 3/10/2026 | Midterm |

# Tentative Syllabus 2

| Week | Date | Material |
|---|---|---|
| 9 | 3/17/2026 | Spring Break- No class |
| 10 | 3/24/2026 | Unit 7. Shared memory and the load-store compute pattern |
| 11 | 3/31/2026 | Unit 8. FIR filters and convolutional layers |
| 12 | 4/7/2026 | Unit 9. Multi-threaded control |
| 13 | 4/14/2026 | Unit 10. Multiple processes and message passing architectures |
| 14 | 4/21/2026 | Buffer |
| 15 | 4/28/2026 | Final Exam |
| | 5/5/2026 | Project presentations |

# Project

- Build your own IP
- You must provide:
  - A clearly defined hardware IP block (RTL or HLS)
  - A well-specified interface and integration model with a host system
  - A testbench and evaluation methodology
  - Documentation that allows others to reproduce your result
- Any topic of choice
  - Computer vision, communications, computing, …
- Work in groups of 2 to 4
  - Individual projects require instructor permission



Host system

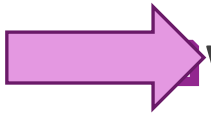
Hardware IP
or accelerator

# Project Grading

- ❑See full rubric on project page on the GitHub pages
- ❑Submission on GitHub
  - Create a repo for the project.  You can start now!
  - Submit only the repo URL.
- ❑IP definition and interface:  Clear role, well-justified boundary between host and IP
- ❑IP implementation:  Efficient design, correct, design choices
- ❑Evaluation:
  - Testbench that are comprehensive, test corner cases, evaluate performance
  - Measure design options
- ❑Process: Graded on GitHub commit history.  No last minute drop before deadline
- ❑Presentation:  Assume you are pitching to a prospective employer

# Outline

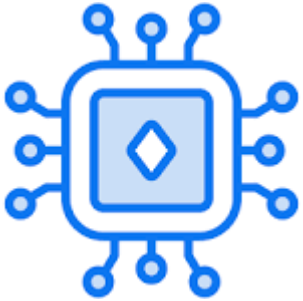
- Course Admin
- What is Custom Hardware and What Can it Do?



# Hardware vs. Software

## Hardware

*Any electronic circuit that performs some computation*



## Software

*Instructions executed on hardware*

# Hardware Comes in Many Forms


General purpose CPU

Intel, AMD, ARM


DSP

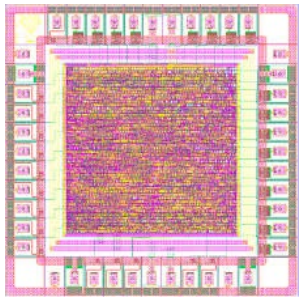
Texas Instruments, Qualcomm Hexagon

GPU

Nvidia
Intel
Radeon


FPGA

Xilinx / AMD,
Intel / Altera



Custom ASIC

Qualcomm,
Apple,
Broadcomm

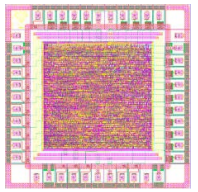
# General Purpose vs. Custom


General purpose CPU

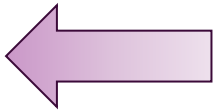

DSP

GPU

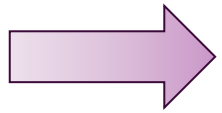

FPGA



Custom ASIC



General purpose,
Applies in many domains,
Re-programmed with software,
Performance via SW optimization, multi-cores

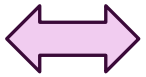


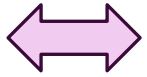
Domain specific,
HW optimized for specific task,
Can enable:

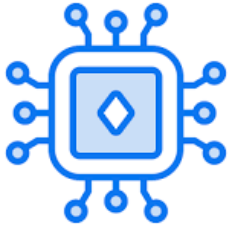
- Higher performance,
- Lower power / OP, size

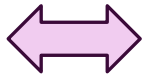
Focus of this course

# Accelerator or IP Model







**Host system**
Runs main application,
User interface,
May be embedded as well
(e.g., smartphone)

**Embedded Processor**
(DSP, ARM,…)
Configures HW,
Manages data transfer,
Performs some DSP

**Hardware IP or accelerator**
Compute intensive tasks
Domain specific

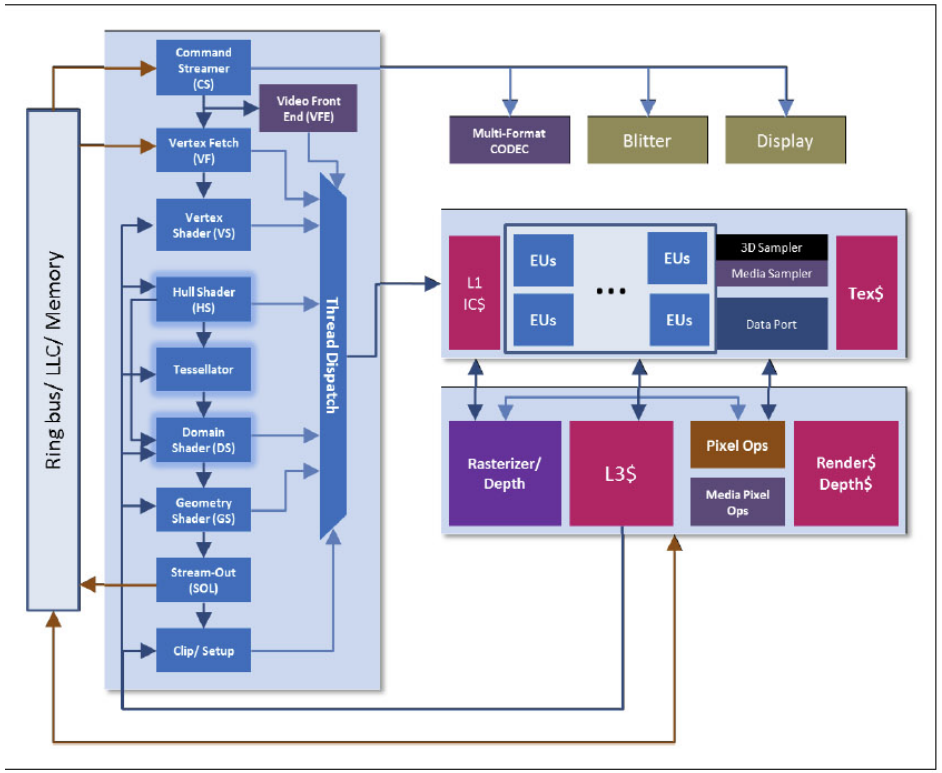
**High-speed IO**
Network interface,
Digital communication data

This class

# What is Custom Hardware Good At

- Compute-intensive kernels
  - High arithmetic density, many operations per data
- Structured, repetitive tasks
  - Same operation repeated many times
  - Regular data flow, memory access
- Regular dataflow and memory patterns
  - Tight control of timing, no OS jitter
- High throughput with low energy
  - Custom datapaths, optimized bit-widths, specialized memories
- Examples: DSP, linear algebra, ML inference, crypto, compression

# Example: Intel Quick Sync Media Processor

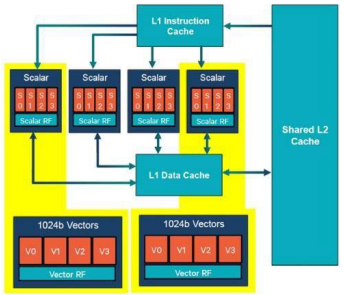



- Intel specialized graphics units
  - Programmable shaders for parallel math
  - Fixed-function blocks (rasterizer, tessellator, texture units)
  - Media engines for video encode/decode (Quick Sync)
- Integrated into Core-i3 / i5 / i7 / i9 processors
  - Physically part of the CPU die
  - Shares memory hierarchy with CPU cores
  - High-speed on-chip interfaces
- Direct access to system memory
  - High-bandwidth internal fabric
  - Connecting CPU, GPU, and media engines
  - Low-latency paths

# Example: 5G Modem Core

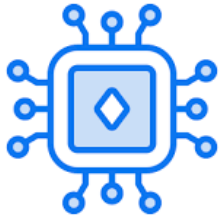

ARM Core

Upper layer SW,
Configuration,
Networking,
Packet processing




Hexagon DSP

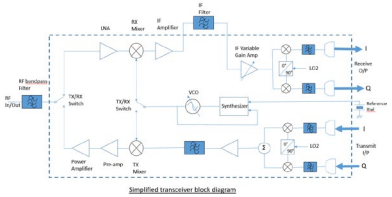
Control channels,
Infrequent, but complex,
data processing



Gsamps / s

5G HW

Filtering,
Equalization,
Decoding,
…

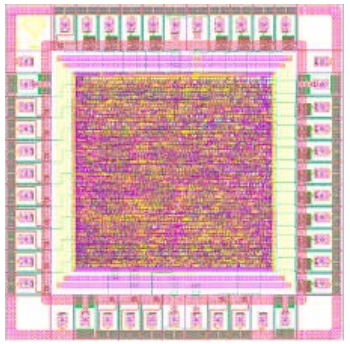


RFIC

Antenna interface,
Radio frequency operations,
DACs / ADCs

# FPGA vs. ASIC


- FPGA: Field-Programmable Gate Array
  - Reconfigurable hardware
  - Great for prototyping and teaching: no fabrication needed
  - Slower and less power-efficient than ASICs
  - Research, rapid development, low-volume products, networking gear, some accelerators
- ASIC: Application-Specific Integrated Circuit
  - Custom hardware chip designed for one purpose (e.g., video codec, 5G modem, ML accelerator)
  - Much faster and more power-efficient than FPGAs
  - Cannot be changed after fabrication — the design is permanent
  - Manufactured ("fabricated") at a semiconductor foundry
  - Ex: TSMC,
  - Smartphones, GPUs, Automotive, …



# Turnaround Time Contrasted

Seconds

Software

Write code → Compile → Run

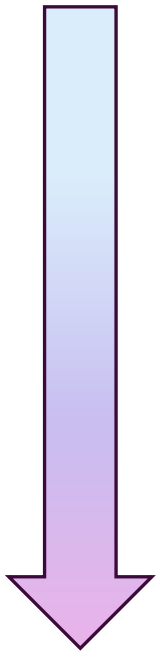
FPGA

Design → Simulate → Synthesize → Generate Bitstream → Run

Custom ASIC

Design → Simulate → Synthesize → Layout → Fabrication

○ ○ ○ → Package → Integrate → Run



Months

# HW Design Is Getting Easier

Traditional view:  HW is Hard

- ❑Rarely-used languages
- ❑Small, elite community
- ❑Complex tool chain
- ❑Long turnaround time
- ❑Need to design concurrent systems

New Tools

- ❑High-Level Synthesis
  - Program in C/C++/Python
- ❑AI-assisted design
  - LLMs are learning HW fast
- ❑Python-based tool flow
  - FPGA Python APIs
  - Python simulation, …

XILINX VITIS

AI

PYNQ

# In-Class Exercise

- ❑ Break into small groups
- ❑ Select any domain of interest to your group
  - Robotics, scientific computing, finance, gaming, …
- ❑ Propose a hardware accelerator for a specific task in that domain
  - Be specific
  - Ex., “Fast solver for molecular simulations”, “Real-time obstacle detection for drones”, …
- ❑ What would it interface with?
  - CPU, Any sensor data / peripherals? How much data does it need?
- ❑ Why would custom hardware help?
  - What are the computations involved? Is it structured, compute intensive?