# ECE 6775
# High-Level Digital Design Automation
# Fall 2024

# Final Project

Cornell University

CSL

# Announcements

- Midterm grades released

- Lab 4 deadline extended to Wednesday Nov 6
  - Useful tips on array partition & reshape posted on Ed
  - Focus on reducing HLS-estimated latency before generating bitstream
  - Designs with high resource utilization may cause routing failures

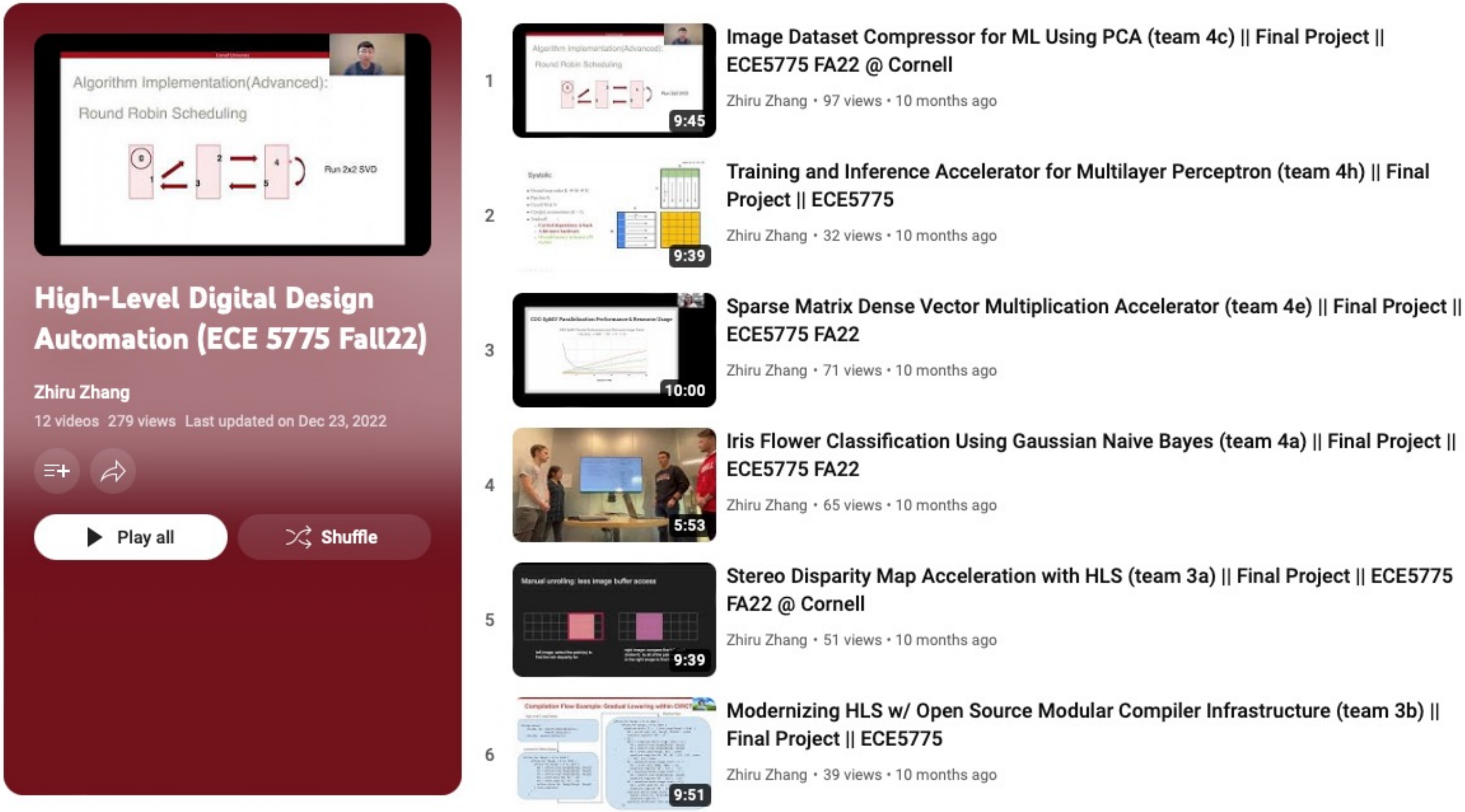
# Project Logistics

- Fill out project teaming sheet today!
  - 3-4 students per teams (12 teams expected)

- Project meetings start on Monday at **Rhodes 471**

- Due dates
  - Project abstract due Friday 11/8 (**no extension**)
  - Presentation due Tuesday 12/10 in a **recorded video**
  - Final report due Friday 12/13

# Project Presentations from 5775 FA22



**High-Level Digital Design Automation (ECE 5775 Fall22)**

Zhiru Zhang

12 videos 279 views Last updated on Dec 23, 2022

1 9:45 Image Dataset Compressor for ML Using PCA (team 4c) || Final Project || ECE5775 FA22 @ Cornell
Zhiru Zhang • 97 views • 10 months ago

2 9:39 Training and Inference Accelerator for Multilayer Perceptron (team 4h) || Final Project || ECE5775
Zhiru Zhang • 32 views • 10 months ago

3 10:00 Sparse Matrix Dense Vector Multiplication Accelerator (team 4e) || Final Project || ECE5775 FA22
Zhiru Zhang • 71 views • 10 months ago

4 5:53 Iris Flower Classification Using Gaussian Naive Bayes (team 4a) || Final Project || ECE5775 FA22
Zhiru Zhang • 65 views • 10 months ago

5 9:39 Stereo Disparity Map Acceleration with HLS (team 3a) || Final Project || ECE5775 FA22 @ Cornell
Zhiru Zhang • 51 views • 10 months ago

6 9:51 Modernizing HLS w/ Open Source Modular Compiler Infrastructure (team 3b) || Final Project || ECE5775
Zhiru Zhang • 39 views • 10 months ago

https://www.youtube.com/playlist?list=PLRvJfry30-22OqHmVYfruWs8Hv-Fnwr57

# Project Abstract (due Friday 11/8)

- Two project themes
  - **App**: Accelerator design for compute/data-intensive applications
  - **Tool**: Compilation/synthesis for accelerator design/programming

- Abstract format
  - Write a concise one-page project overview consisting of 2-3 paragraphs
  - Include the project title, theme, and list of team members
  - Summarize the project, outlining key approaches
  - Justify the project's feasibility within the given time constraints

# Theme 1 (App)
# Application-Specific Accelerator Design

- Utilize HLS to create FPGA-based hardware accelerators for compute-intensive applications
  - Explore hardware customization techniques in emerging application domains, e.g., computer vision, genomics, machine learning, confidential computing.

- Design languages: C++ or Python DSL

- HLS tools: AMD Xilinx Vivado/Vitis HLS

- FPGA platforms
  - ZedBoard: a small device; relatively short compile time
  - Alveo (datacenter): a much larger device equipped with high-bandwidth memory (HBM); long compile time (~hours)

# How to Choose an Application?

**Ideal application characteristics for hardware acceleration**
**(a)** Abundant parallelism
**(b)** Custom (low-bitwidth) numeric types
**(c)** Distributed memory accesses

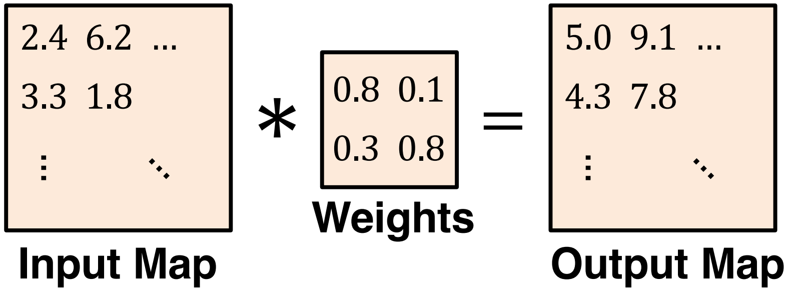
Lab 1 CORDIC: **(b)**

Lab 2/3 K-Nearest Neighbors: **(a) (c)**

Lab 4 Binary Neural Network: **(a) (b) (c)**

# Lab 4: Extreme Quantization with Binarization

## CNN



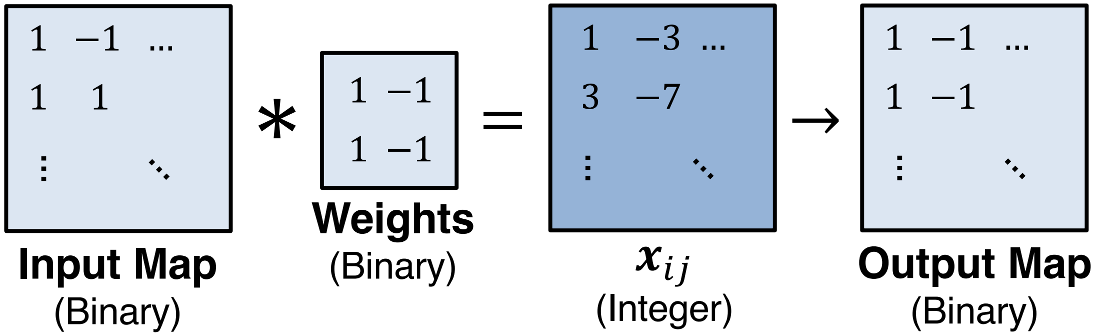
$$\begin{bmatrix}2.4 & 6.2 & \ldots\\ 3.3 & 1.8 & \\ \vdots & & \ddots\end{bmatrix} * \begin{bmatrix}0.8 & 0.1\\ 0.3 & 0.8\end{bmatrix} = \begin{bmatrix}5.0 & 9.1 & \ldots\\ 4.3 & 7.8 & \\ \vdots & & \ddots\end{bmatrix}$$

Input Map

Weights

Output Map

## BNN



$$\begin{bmatrix}1 & -1 & \ldots\\ 1 & 1 & \\ \vdots & & \ddots\end{bmatrix} * \begin{bmatrix}1 & -1\\ 1 & -1\end{bmatrix} = \begin{bmatrix}1 & -3 & \ldots\\ 3 & -7 & \\ \vdots & & \ddots\end{bmatrix} \rightarrow \begin{bmatrix}1 & -1 & \ldots\\ 1 & -1 & \\ \vdots & & \ddots\end{bmatrix}$$

Input Map (Binary)

Weights (Binary)

$x_{ij}$ (Integer)

Output Map (Binary)

**Key Differences**

1. Inputs are binarized (–1 or +1)
2. Weights are binarized (–1 or +1)
3. MAC becomes XNOR+Popcount

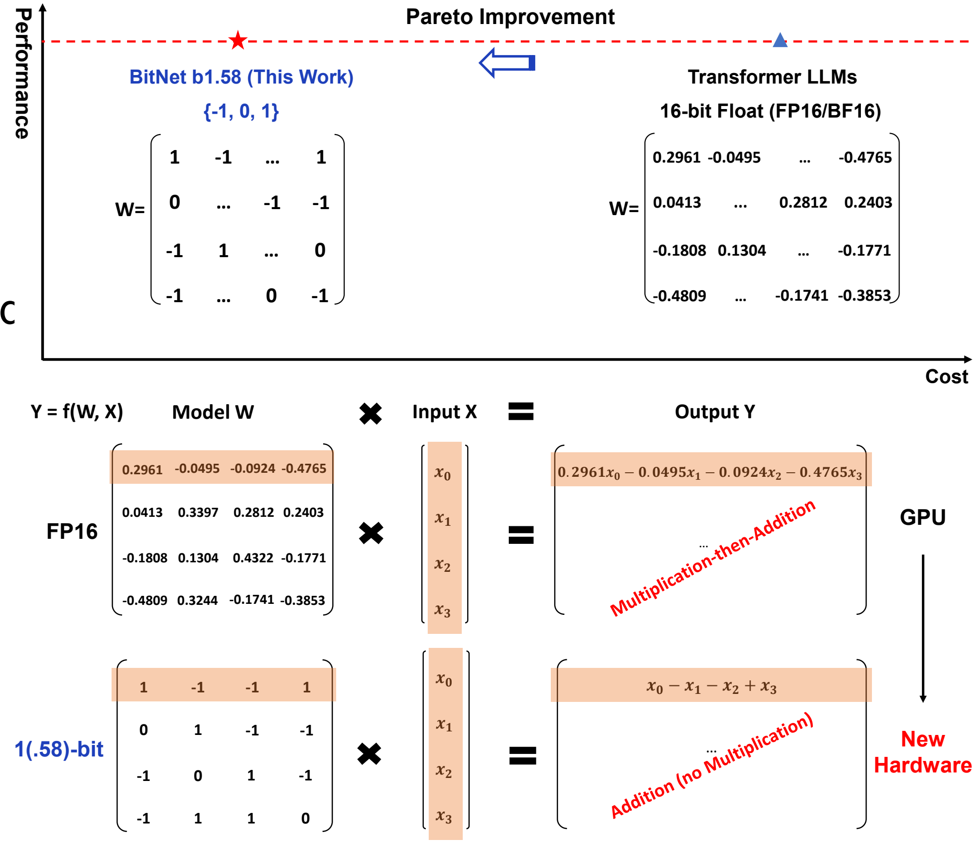
BNNs are well suited for FPGAs (rich in LUTs)

# Theme 1: A Potential (Umbrella) Project

- BitNet: LLM with ternary weights {-1, 0, 1}
  - No multiplication required for matrix multiplication
  - C++ implementation open sourcec on GitHub

- Implement key kernels in HLS (e.g., attention, feedforward network)

- Teams can focus on a single kernel or choose to implement multiple kernels



$$W=\begin{pmatrix}1 & -1 & \dots & 1\\ 0 & \dots & -1 & -1\\ -1 & 1 & \dots & 0\\ -1 & \dots & 0 & -1\end{pmatrix}$$

$$W=\begin{pmatrix}0.2961 & -0.0495 & \dots & -0.4765\\ 0.0413 & \dots & 0.2812 & 0.2403\\ -0.1808 & 0.1304 & \dots & -0.1771\\ -0.4809 & \dots & -0.1741 & -0.3853\end{pmatrix}$$

$$\begin{pmatrix}0.2961 & -0.0495 & -0.0924 & -0.4765\\ 0.0413 & 0.3397 & 0.2812 & 0.2403\\ -0.1808 & 0.1304 & 0.4322 & -0.1771\\ -0.4809 & 0.3244 & -0.1741 & -0.3853\end{pmatrix} \times \begin{pmatrix}x_0\\ x_1\\ x_2\\ x_3\end{pmatrix} = \begin{pmatrix}0.2961x_0 - 0.0495x_1 - 0.0924x_2 - 0.4765x_3\\ \dots\end{pmatrix}$$

$$\begin{pmatrix}1 & -1 & -1 & 1\\ 0 & 1 & -1 & -1\\ -1 & 0 & 1 & -1\\ -1 & 1 & 1 & 0\end{pmatrix} \times \begin{pmatrix}x_0\\ x_1\\ x_2\\ x_3\end{pmatrix} = \begin{pmatrix}x_0 - x_1 - x_2 + x_3\\ \dots\end{pmatrix}$$

https://github.com/microsoft/BitNet

S. Ma et al., The Era of 1-bit LLMs: All Large Language Models are in 1.58 Bits. arXiv e-print 2024.

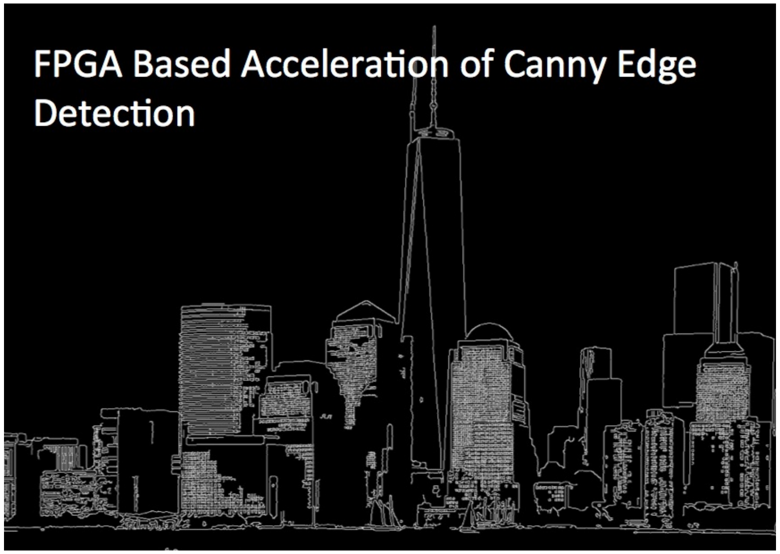
# Theme 1: Other Topics to Consider

- Compute customization
  - Systolic arrays for dense linear algebra, e.g., MV, MM
    - Evaluate the design using a simple ML model such as an attention layer in Transformer

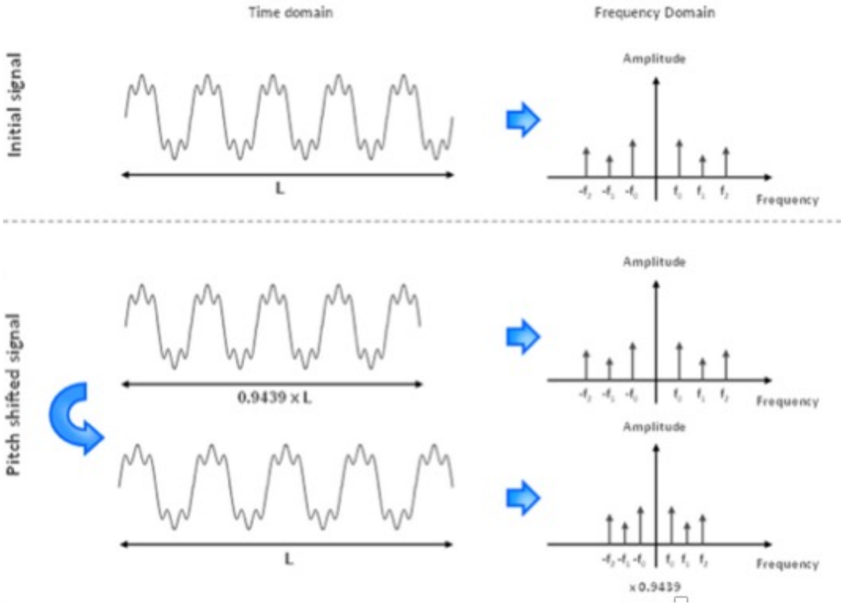
- Data type customization
  - HLS library for low-precision floating point arithmetic
    - Evaluate the library using lab designs such as CORDIC
  - Large-bitwidth integer arithmetic on FPGAs (e.g., 512b multiply)
    - Evaluate the design using a crypto application

- Memory customization
  - Reuse buffer for stencil-based image/video processing
  - Custom memory layout for sparse linear algebra

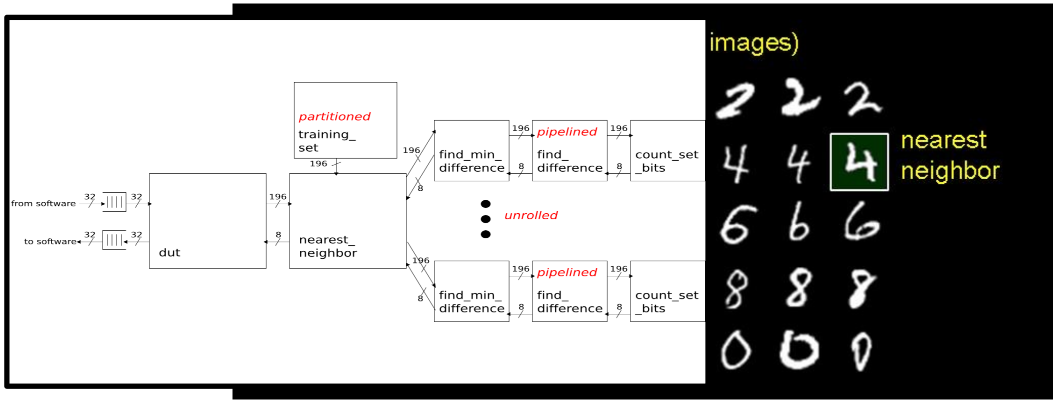
# Theme 1: Examples from Previous Years



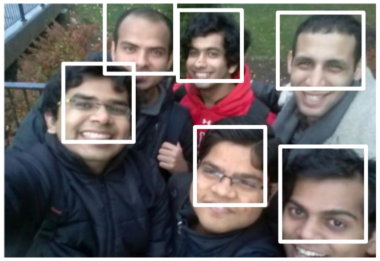
Canny Edge Detection



Real-Time Vocal Processor of Pop Music



Digit Recognition



Face Detection

# Theme 1: Do’s and Don’ts

- Choose an application you are familiar with (or one that’s easy to grasp)

- Minimize “setup” time for the baseline implementation (under 1 week)

- Analyze parallelism and OI before HLS coding

- Focus on HLS-level performance optimization and minimize runs of bitstream generation

# Anticipated Project Schedule

- Week 1 (Nov 4): Brainstorm and project abstract
- Week 2 (Nov 11): Complete baseline design with proper testing
- Week 3 (Nov 18): Perform design optimizations at HLS level
- Week 4 (Nov 25): Implement design on board
- Week 5 (Dec 2): Continue optimization and work on the project report

# Theme 2 (Tool)
# Accelerator-Centric Compilation/Synthesis

- Develop new compilation and/or HLS techniques
  - Compiler analysis & transformations for accelerators
  - Improving core HLS algorithms: scheduling, pipelining, binding
  - Optimizing new design metrics (e.g., security)

- Software frameworks
  - Open-source compiler infrastructure: LLVM, MLIR, Allo
  - Commercial HLS as a back end: Vivado/Vitis HLS

# Theme 2: Topics to Consider

- Machine learning (ML) for HLS
  - New scheduling algorithms leveraging ML techniques
  - ML for latency and resource estimation
  - HLS optimization on GPUs

- Automated performance modeling
  - Compiler analysis passes to estimate OI

- HLS for ASIC design
  - Extend an open-source MLIR based HLS toolflow

# Theme 2 Example: Trace-Based Array Banking

| Cycles | RD0 | | RD1 | | RD2 | | RD3 | |
|---|---|---|---|---|---|---|---|---|
| | i | j | i | j | i | j | i | j |
| 1 | 0000 | 0000 | 0000 | 0010 | 0010 | 0000 | 0010 | 0010 |
| 2 | 0000 | 0001 | 0000 | 0011 | 0010 | 0001 | 0010 | 0011 |
| … | … | … | … | … | … | … | … | … |
| 10 | 0000 | 1001 | 0000 | 1011 | 0010 | 1001 | 0010 | 1011 |
| … | … | … | … | … | … | … | … | … |

```
int A[Rows][Cols];
int sum;

for ( int i = 1; i < Rows – 1; i ++ )
   for ( int j = 1; j < Cols – 1; j ++ )
        sum =  A[i-1][j-1] + A[i-1][j+1]
                  + A[i+1][j-1] + A[i+1][j+1];
```

**Important bits (mask)**

| | | | | | |
|---|---|---|---|---|---|
| i | … | $i_3$ | $i_2$ | $i_1$ | $i_0$ |
| j | … | $j_3$ | $j_2$ | $j_1$ | $j_0$ |

**Hash function**

| Mask | Bank |
|---|---|
| 00 | 0 |
| 01 | 1 |
| 10 | 2 |
| 11 | 3 |

| i/j | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 2 | 2 | 2 | 3 | 3 | 2 | 2 | 3 | 3 |
| 3 | 2 | 2 | 3 | 3 | 2 | 2 | 3 | 3 |
| 4 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 5 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 6 | 2 | 2 | 3 | 3 | 2 | 2 | 3 | 3 |
| 7 | 2 | 2 | 3 | 3 | 2 | 2 | 3 | 3 |

Partitioning array A

# Theme 2: Do’s and Don’ts

- Leverage open-source compiler infrastructures and programming frameworks
  - Avoid building a new IR from scratch

- Formulate the problem in an exact way before implementing any heuristic algorithms

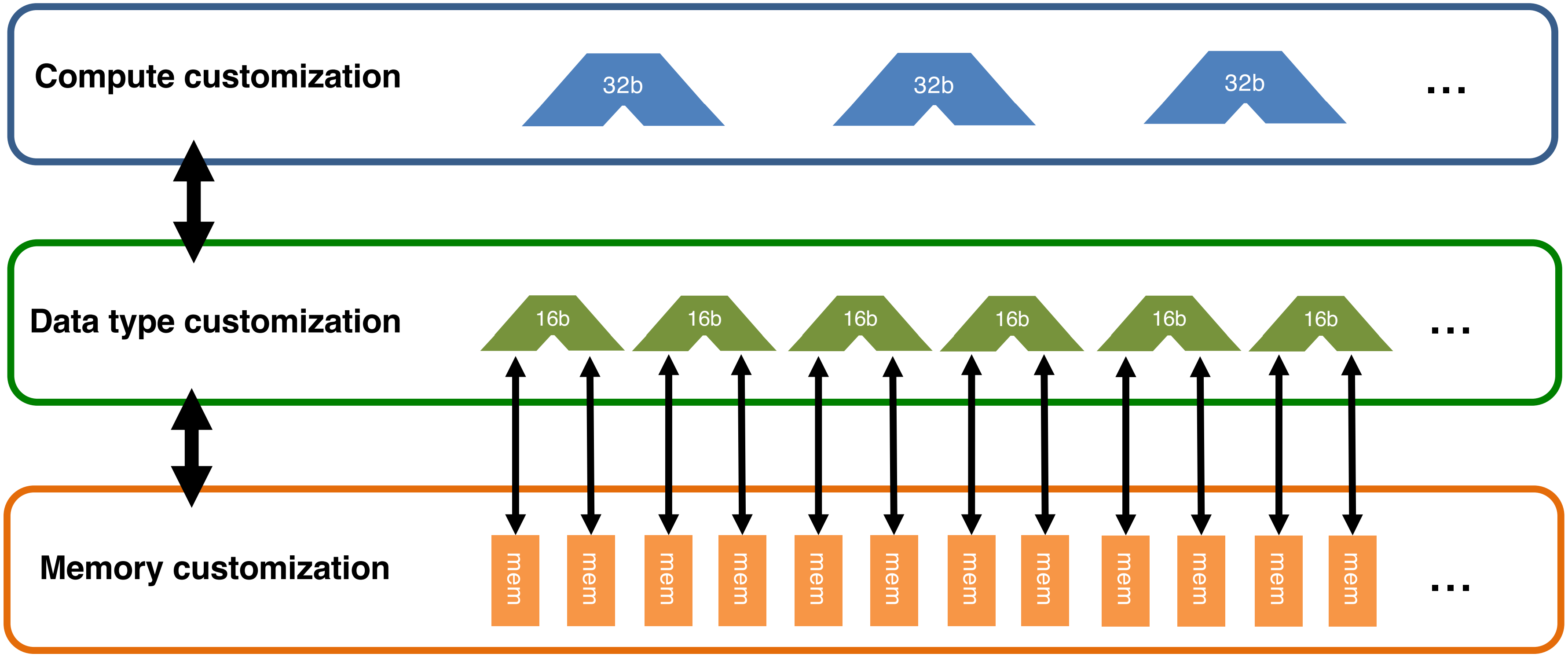
# Recap: Learning Outcomes (The Intangibles)

- Develop a principled approach to analyzing accelerator design process and essential design factors (e.g., parallelism, resources, precision)

- Gain comprehensive insights into accelerator design from the perspective of an HLS compiler

We aim to achieve these objectives through **a balanced mix of theoretical foundations** (lectures & homework) **and practical applications** (labs & project)

# Essentials of Hardware Specialization



**Parallelism** **Resources** **Data**

Pipelining Parallel processing Bandwidth Reuse Precision Layout

# Recap: What this Course is About

*Hardware/Software* ***Co-Design***

- Specify applications/algorithms in software programs
- Synthesize software descriptions into special-purpose hardware components, namely, *accelerators*
  - Perform manual source-level code optimizations
  - Utilize automatic compilation & synthesis optimizations
  - Explore performance-cost trade-offs
- Realize the synthesized accelerators on FPGAs

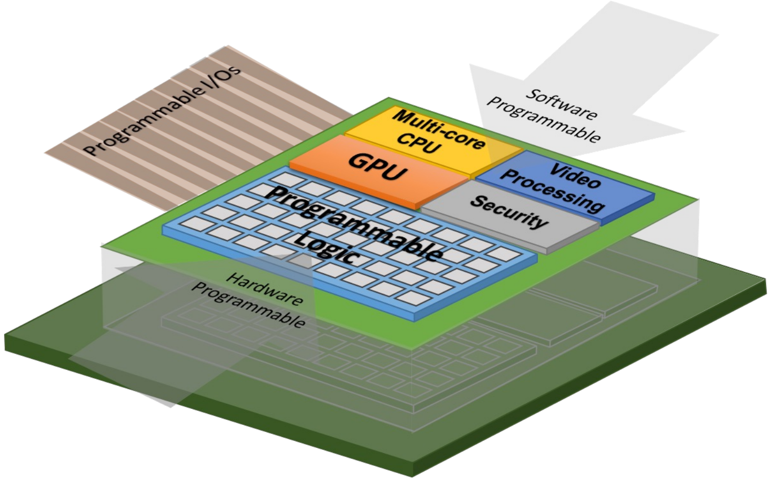
# Co-Design Revisited

**Mapping**

“HARD” **Software**

**to**

“SOFT” **Hardware**



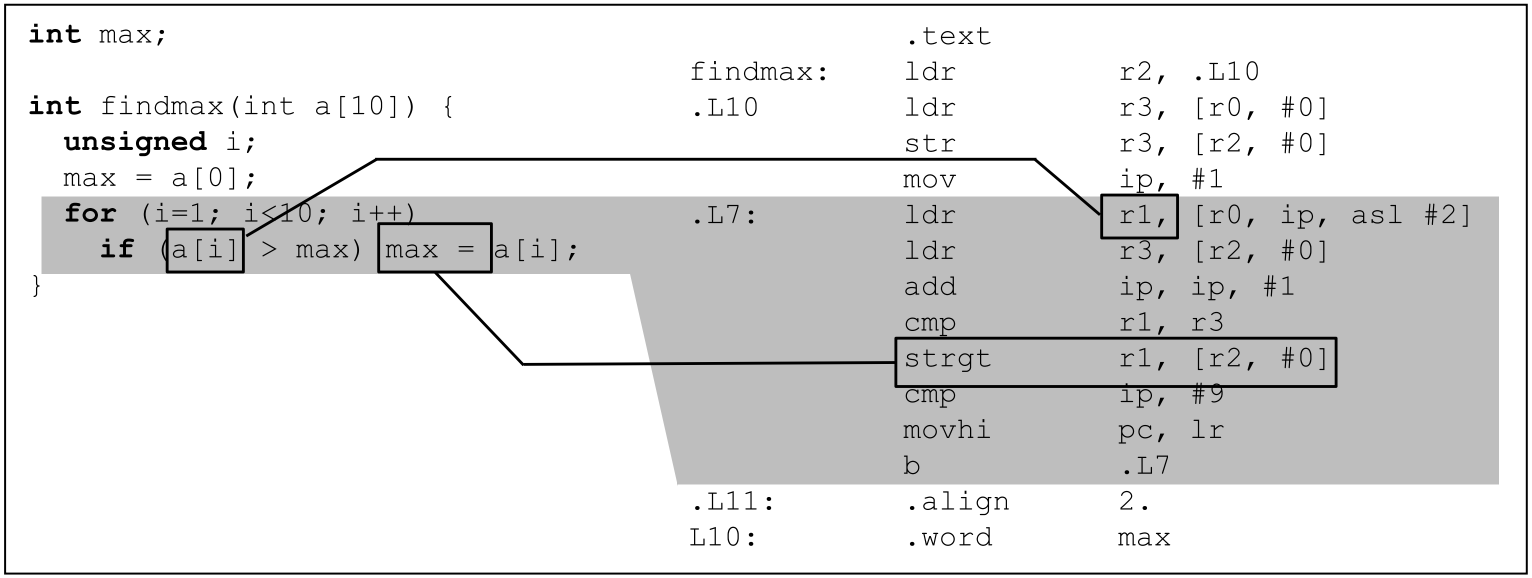
“SOFT”: FPGA is a reconfigurable fabric

“HARD”: Performance-oriented programming is challenging, esp. for FPGAs

# Blurred Line Between Hardware and Software Design

- Hardware

- Software



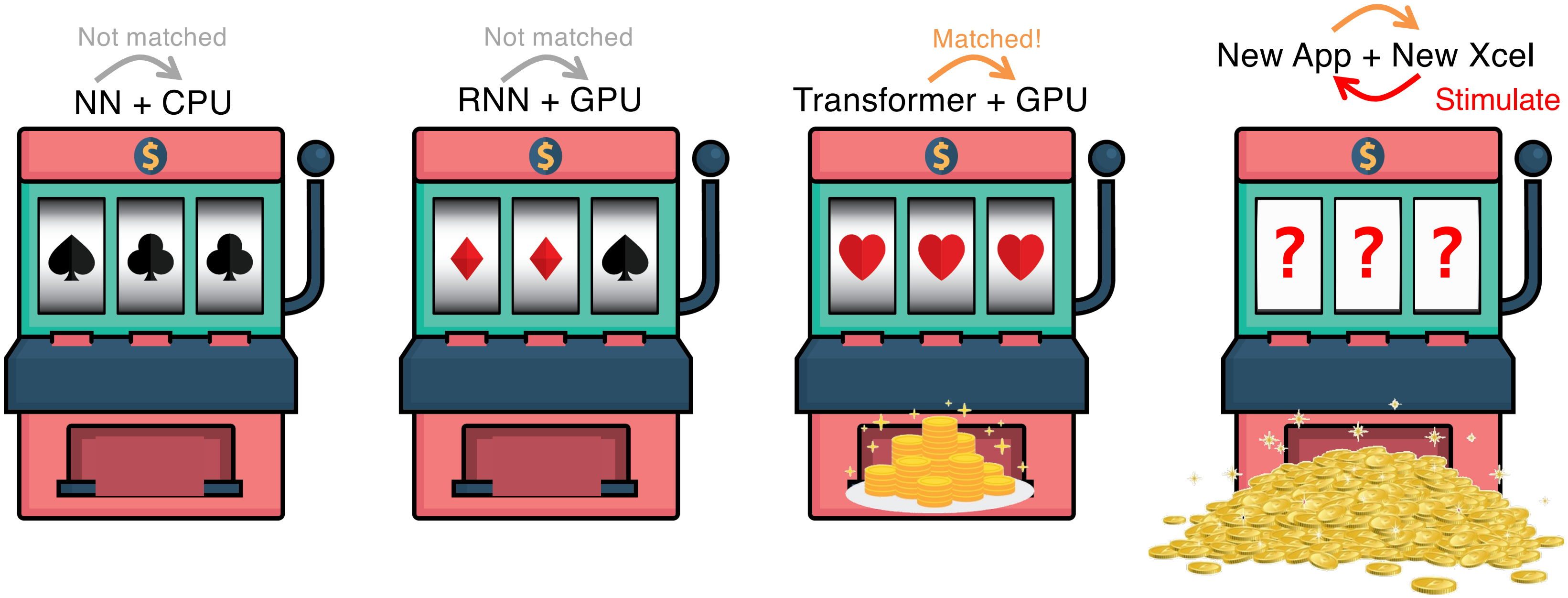
```
int max;

int findmax(int a[10]) {
  unsigned i;
  max = a[0];
  for (i=1; i<10; i++)
    if (a[i] > max) max = a[i];
}
```

```
            .text
findmax:    ldr      r2, .L10
.L10        ldr      r3, [r0, #0]
            str      r3, [r2, #0]
            mov      ip, #1
.L7:        ldr      r1, [r0, ip, asl #2]
            ldr      r3, [r2, #0]
            add      ip, ip, #1
            cmp      r1, r3
            strgt    r1, [r2, #0]
            cmp      ip, #9
            movhi    pc, lr
            b        .L7
.L11:       .align   2.
L10:        .word    max
```

# Hardware Lottery* => Jackpot!

- High performance is achieved only when the **application characteristics match** the underlying **hardware architecture**



Not matched

NN + CPU

Not matched

RNN + GPU

Matched!

Transformer + GPU

New App + New Xcel

Stimulate

* Hardware Lottery: https://hardwarelottery.github.io/

## Next Time

- Project meetings (Rhodes 471)