ECE 6775
High-Level Digital Design Automation
Fall 2023

HLS Design Practice
Midterm Review

Cornell University

CSL

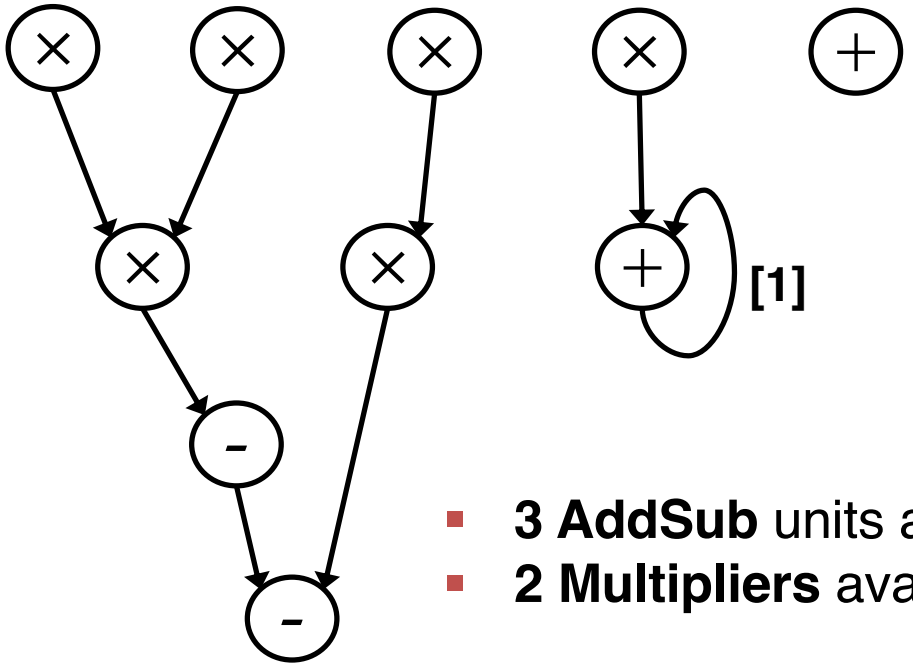
Announcements

- Midterm on Thursday at **8:30am**

- Another reading assignment
  - C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks", FPGA 2015
  - **Complete reading before Tuesday 10/24**

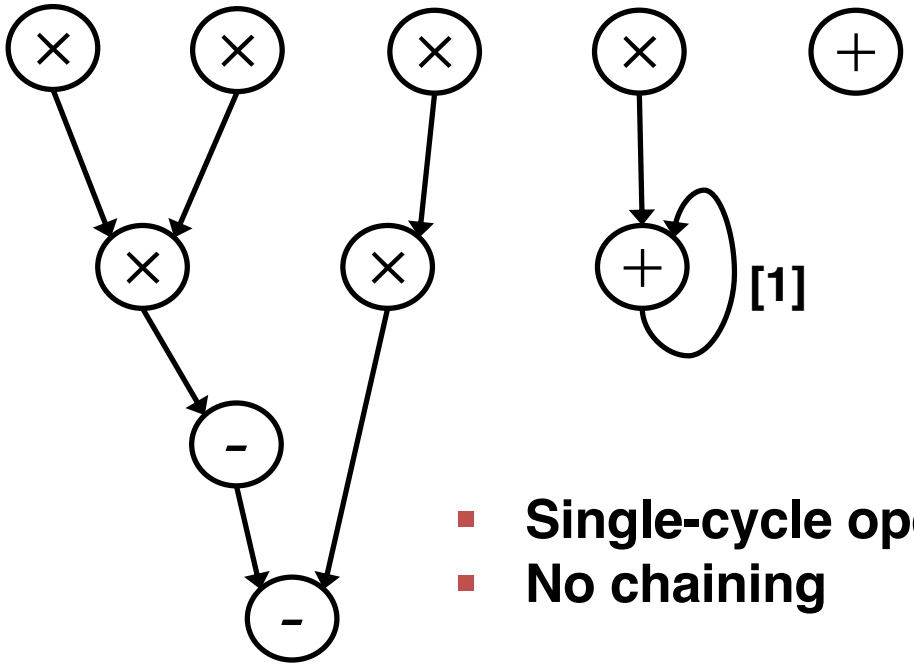
- Lab 4 will be released today

Example: What's the ResMII



- **3 AddSub** units available
- **2 Multipliers** available

Example: What's the RecMII



- Single-cycle operations
- No chaining

Case Study: CORDIC

```
void cordic(theta_type theta, cos_sin_type &s, cos_sin_type &c) {
  double K_const = 0.607252935008881256169 4;
  theta_type current = 0;
  cos_sin_type X = K_const; Y = 0, T;
  for (int step = 0; step < 20; step++) {
    if (theta > current) {
      T = X - (Y >> step);
      Y = (X >> step) + Y;
      X = T;
      current = current + cordic_ctab[step];
    } else {
      T = X + (Y >> step);
      Y = -(X >> step) + Y;
      X = T;
      current = current - cordic_ctab[step];
    }
  }
  s = Y;
  c = X;
}
```

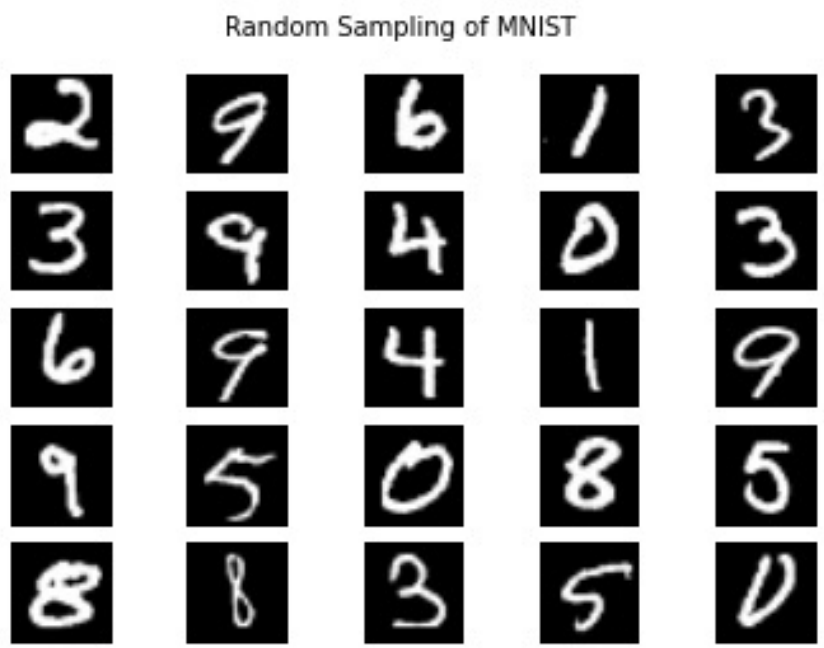
Pipeline the whole function (loop inside automatically unrolled)

II=1 means one CORDIC per cycle

Unroll or pipeline this loop?

Case Study: Digit Recognition

- Use a simple machine learning algorithm to recognize handwritten digits
  - 2000 training instances per digit
  - Each training/test instance is a 7x7 bitmap after downsampling



MNIST dataset: http://yann.lecun.com/exdb/mnist/

K-Nearest-Neighbor (KNN) Implementation

```
bit4 digitrec( digit input )
{
  #include "training_data.h"
  // This array stores K minimum distances per training set
  bit6 knn_set[10][K_CONST];
  // Initialize the knn set
  for ( int i = 0; i < 10; ++i )
    for ( int k = 0; k < K_CONST; ++k )
      // Note that the max distance is 49
      knn_set[i][k] = 50;
```

Main compute loop

```
L2000: for ( int i = 0; i < TRAINING_SIZE; ++i ) {
  L10:   for ( int j = 0; j < 10; j++ ) {

    // Read a new instance from the training set
    digit training_instance = training_data[j * TRAINING_SIZE + i];
    // Update the KNN set
    update_knn( input, training_instance, knn_set[j] );
  }
}
```

Assuming 10 cycles per innermost loop (L10)

~200K cycles by default without optimizations

10x Speedup through Parallel Processing

```
bit4 digitrec( digit input )
{
  #include "training_data.h"
  // This array stores K minimum distances per training set
  bit6 knn_set[10][K_CONST];
  // Initialize the knn set
  for ( int i = 0; i < 10; ++i )
    for ( int k = 0; k < K_CONST; ++k )
      // Note that the max distance is 49
      knn_set[i][k] = 50;
```

Unroll inner loop completely

```
  L2000: for ( int i = 0; i < TRAINING_SIZE; ++i ) {
   L10:    for ( int j = 0; j < 10; j++ ) {

      // Read a new instance from the training set
      digit training_instance = training_data[j * TRAINING_SIZE + i];
      // Update the KNN set
      update_knn( input, training_instance, knn_set[j] );
    }
  }
```

Partition training set into 10 banks

10 instances of "update_knn" running in parallel

~20K cycles after parallelization

Further Speedup through Pipelining

```
bit4 digitrec( digit input )
{
  #include "training_data.h"
  // This array stores K minimum distances per training set
  bit6 knn_set[10][K_CONST];
  // Initialize the knn set
  for ( int i = 0; i < 10; ++i )
    for ( int k = 0; k < K_CONST; ++k )
      // Note that the max distance is 49
      knn_set[i][k] = 50;

  L2000: for ( int i = 0; i < TRAINING_SIZE; ++i ) {
    L10:   for ( int j = 0; j < 10; j++ ) {

      // Read a new instance from the training set
      digit training_instance = training_data[j * TRAINING_SIZE + i];
      // Update the KNN set
      update_knn( input, training_instance, knn_set[j] );
    }
  }
```

Pipeline outer loop

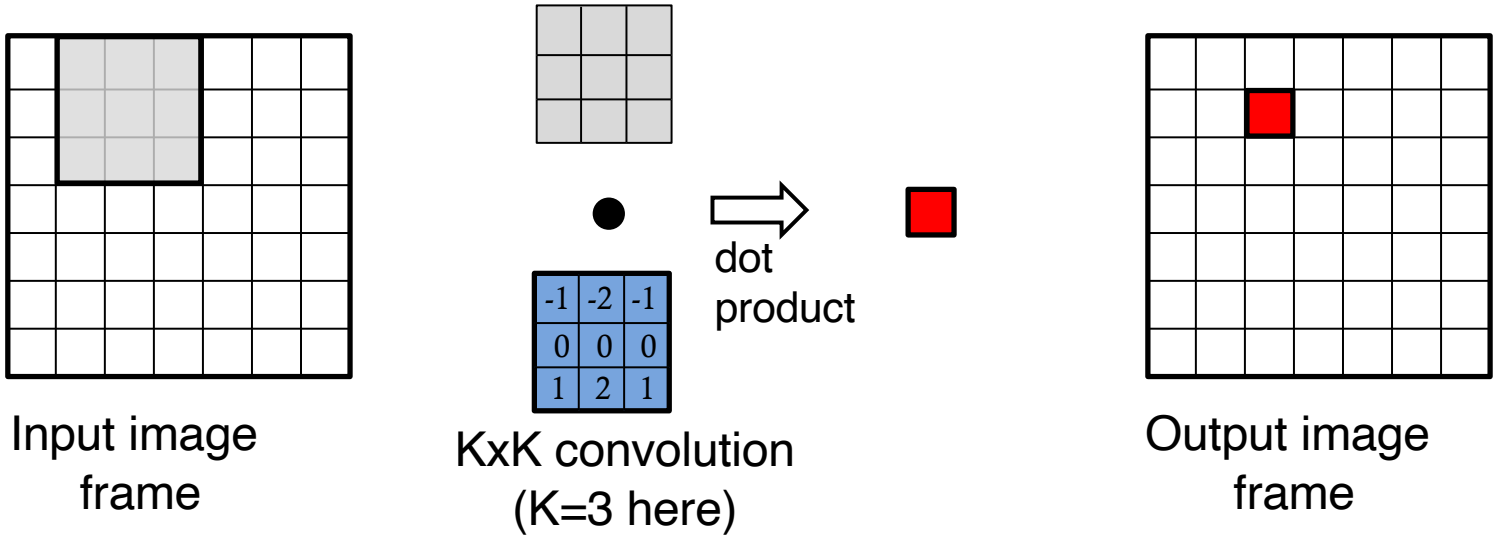
Outer loop (L2000) pipelined to II=1

~2K cycles after pipelining

Case Study: Convolution for Image Processing

- **Convolution** is pervasive in image/video processing and ML – performed over overlapping windows (aka stencils)

$$(Img \otimes f)_{\left[n+\frac{k-1}{2},m+\frac{k-1}{2}\right]} = \sum_{i=0}^{k-1}\sum_{j=0}^{k-1} Img_{[n+i][m+j]} \cdot f_{[i,j]}$$



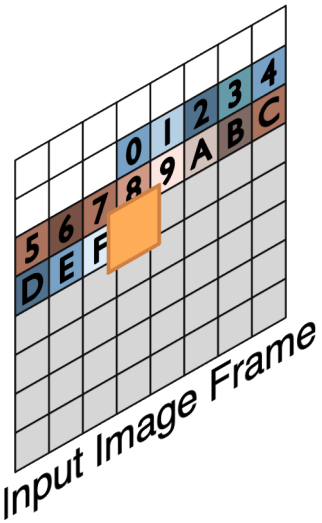
Pipelining 3x3 Convolution

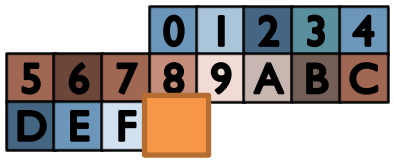
```
for (r = 1; r < H; r++)
    for (c = 1; c < W; c++) {
      #pragma HLS pipeline II=?
      for (i = 0; i  < 3; i++)
        for (j = 0; j < 3; j++)
          out[r][c] += img[r+i-1][c+j-1] * f[i][j];
     }
```

- Inner loops “i" & j are automatically unrolled
- The 3x3 filter array “f” is partitioned into 9 registers
- The entire input image “img” is stored in an on-chip buffer with **two read ports**

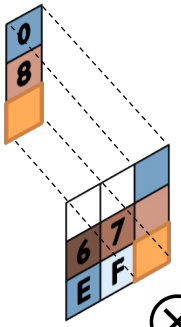
ResMII = ? What about RecMII?

Achieving II=1 for 3x3 Convolution using a Line Buffer and Shift Registers





Pixels in **line buffer**
(stores 2 lines using on-chip SRAMs)



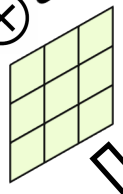
1. **Push 3 pixels into shift registers** –
   1 new pixel +
   2 from line buffer

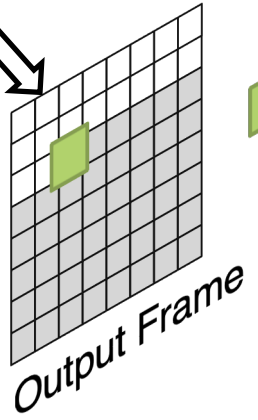
2. **Update line buffer**
   by removing the oldest pixel and shifting in the new one

Shift registers

⊗ *Convolution (conv)*

Filter weights





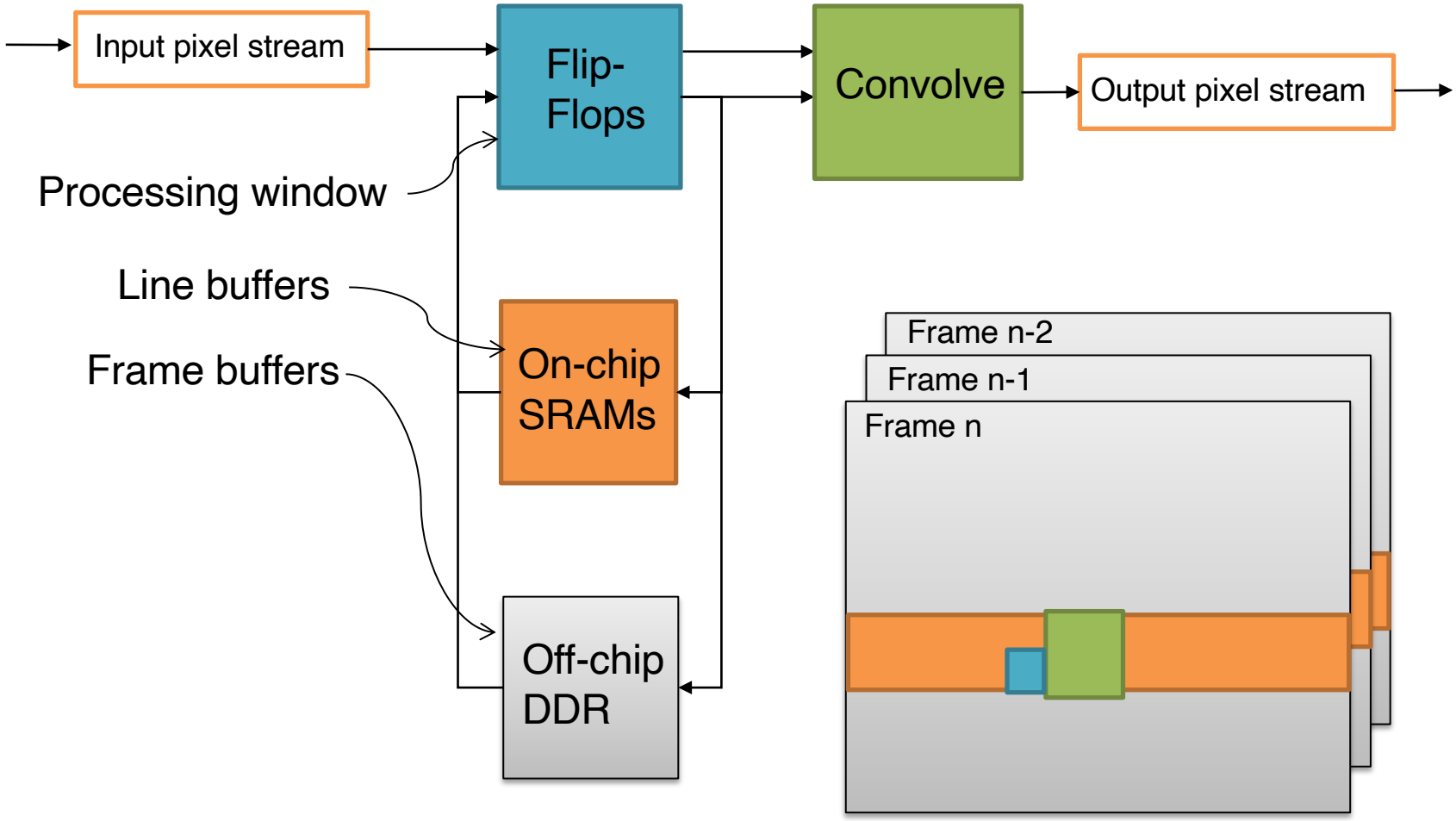
Output pixel produced by conv

New pixel read from frame buffer in main memory (DRAM)

Line Buffer + Shift Registers:
a custom “cache” + a custom “register file”

Resulting Specialized Memory Hierarchy

- Memory architecture customized for convolution



HLS Code Snippet

```
1   LineBuffer<2,C,pixel_t> linebuf;
2   Window<3,3,pixel_t> window;
3   for (int r = 1; r < R+1; r++) {
4     for (int c = 1; c < C+1; c++) {
5       #pragma HLS pipeline II=1
6       pixel_t new_pixel = img[r][c];
7       // Update shift window
8       window.shift_left();
9       if (r < R && c < C) {
10        for (int i = 0; i < 2; i++ )
11          window.insert(buf[i][c]);
12      }
13      else { // zero padding
14        for (int i = 0; i < 2; i++)
15          window.insert(0);
16      }
17      window.insert(new_pixel);
18      // Update line buffer
19      linebuf.shift_up(c);
20      if (r < R && c < C)
21        linebuf[1].insert(c, new_pixel);
22      else // Zero padding
23        linebuf[1].insert(c, 0);
24      // Perform 3x3 convolution
25      out[r-1][c-1] = convolve(window, weights);
26    }
27  }
```

Array Partitioning Caveats

Example 1: Array partitioning through *constant indices* (after unrolling)

Original code

```
for (int i = 0; i < N; ++i)
  for (int k = 0; k < 8; ++k)
    sum += A[k][i];
```

partition "A" to 8 sub-arrays on dimension "k"
without unrolling the inner loop

Transformed loop body

```
switch (k) {
  case 0: sum += A_0[i]
  case 1: sum += A_1[i]
  ...
  case 7: sum += A_7[i]
}
```

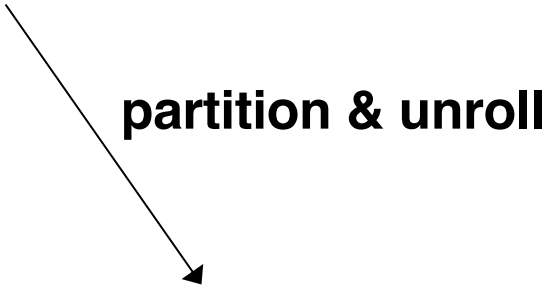
Inefficient design
The switch-case statement will be synthesized into (large) multiplexers in RTL

Array Partitioning Caveats

Example 1: Array partitioning through *constant indices* (after unrolling)

Original code

```
for (int i = 0; i < N; ++i)
  for (int k = 0; k < 8; ++k)
    sum += A[k][i];
```



Transformed code

```
for (int i = 0; i < N; ++i) {
  sum += A_0[i];
  sum += A_1[i];
  ...
  sum += A_7[i];
}
```

Efficient design

After unrolling inner loop, the resulting indices on the "k" dimension become constant, which simplify the logic after array partitioning

Midterm

- Topics covered
  - Hardware specialization
  - Algorithm basics
  - FPGA
  - C-based synthesis
  - Control flow graph and SSA
  - Scheduling
  - Resource sharing
  - Pipelining

Key Topics (1)

- Algorithm basics
  - Time complexity, esp. big-O notation
  - Graphs
    - Trees, DAGs, topological sort
    - BDDs, timing analysis
- FPGAs
  - LUTs and LUT mapping
- C-based synthesis
  - Arbitrary precision and fixed-point types
  - Key HLS optimizations to improve design performance

Key Topics (2)

- Control data flow graph
  - Dominance relation
  - Loops
  - SSA

- Scheduling
  - TCS & RCS algorithms: ILP, list scheduling, SDC
    - Operation chaining, frequency/latency/resource constraints
  - Ability to devise a simple scheduling algorithm to optimize a design metric

Key Topics (3)

- Resource sharing
  - Conflict and compatibility graphs
  - Ability to determine minimum resource usage in # of functional units and/or registers, given a fixed schedule

- Pipelining
  - Dependence types
  - Ability to determine minimum II given a code snippet
    - Modulo scheduling concepts: MII, RecMII, ResMII

Next Lecture

- DNN Acceleration on FPGAs
  - Complete the reading assignment by Tuesday