ECE 6775
High-Level Digital Design Automation
Fall 2023

HLS Design Practice
Midterm Review
Announcements

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

- Another reading assignment
  - 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

```c
void cordin(theta_type theta, cos_sin_type &s, cos_sin_type &c) {
    double K_const = 0.6072529350088812561694;
    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 + cordin_cstab[step];
        } else {
            T = X + (Y >> step);
            Y = -(X >> step) + Y;
            X = T;
            current = current - cordin_cstab[step];
        }
    }

    s = Y;
    c = X;
}
```

Unroll or pipeline this loop?

Pipeline the whole function (loop inside automatically unrolled)

II=1 means one CORDIC per cycle
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

```c
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] );
  }
}
```

Main compute loop

Assuming 10 cycles per innermost loop (L10)
~200K cycles by default without optimizations
10x Speedup through Parallel Processing

```c
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] );
            }
        }
}
```

Unroll inner loop completely
Partition training set into 10 banks

10 instances of “update_knn” running in parallel
~20K cycles after parallelization
Further Speedup through Pipelining

```c
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] );
          }
}
```

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)

\[
(Lmg \otimes f)_{n+\frac{k-1}{2},m+\frac{k-1}{2}} = \sum_{i=0}^{k-1} \sum_{j=0}^{k-1} Lmg_{n+i,m+j} \cdot f_{i,j}
\]

Input image frame  
KxK convolution (K=3 here)  
Output image frame
Pipelining 3x3 Convolution

```c
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

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

Line Buffer + Shift Registers: a custom “cache” + a custom “register file”
Resulting Specialized Memory Hierarchy

- Memory architecture customized for convolution

Diagram:
- Input pixel stream
- Flip-Flops
- Convolve
- Output pixel stream

Processing window
- Line buffers
- Frame buffers
- On-chip SRAMs
- Off-chip DDR

Frames:
- Frame n
- Frame n-1
- Frame n-2
The major compute optimization for Optical Flow is the optimization to overlap different stages of the image processing pipeline. For the dataflow in loop rather than in a function. This is implemented using C++.

```
void optical_flow(pixel_t frame1[MAX_HEIGHT * MAX_WIDTH],
                  pixel_t frame2[MAX_HEIGHT * MAX_WIDTH],
                  pixel_t frame3[MAX_HEIGHT * MAX_WIDTH],
                  pixel_t frame4[MAX_HEIGHT * MAX_WIDTH],
                  pixel_t frame5[MAX_HEIGHT * MAX_WIDTH],
                  velocity_t outputs[MAX_HEIGHT * MAX_WIDTH])
```

The Optical Flow application captures the motion pattern from each frame. Lines 5-10 shows the unoptimized version signature of the top-level hardware function with and without a view of the code.

```
// Perform 3x3 convolution
out[r-1][c-1] = convolve(window, weights);
```
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 loop body
```
switch (k) {
  case 0: sum += A_0[i]
  case 1: sum += A_1[i]
  ...
  case 7: sum += A_7[i]
}
```

*partition “A” to 8 sub-arrays on dimension “k” without unrolling* the inner loop

**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:
```c
for (int i = 0; i < N; ++i) {
    sum += A[k][i];
}
```

Transformed code:
```c
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