Fixed-Point Types
Analysis of Algorithms
Announcements

- Lab 1 on CORDIC is released
  - Due Monday 9/10 @11:59am

- Part-time PhD TA: Hanchen Jin (hj424)
  - Office hour: Mondays 11:00am-12:00pm @ Rhodes 312
Outline

- More on FPGA-based computing
  - Customized memory architecture
    - Case study on convolution
  - Customized data types
    - Arbitrary precision integer and fixed-point types

- Basics of algorithm analysis
  - Complexity analysis and asymptotic notations
  - Taxonomy of algorithms
Example: Implementing Logic with LUTs

(1) How many 3-input LUTs are needed to implement the following full adder?
(2) How about using 4-input LUTs?

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C_{in}</th>
<th>C_{out}</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Recap: FPGA as a Programmable Accelerator

- Massive amount of fine-grained parallelism
  - Highly parallel/deeply pipelined architecture
  - Distributed data/control dispatch
- Silicon configurable to fit the application
  - Compute the exact algorithm at desired numerical accuracy
  - Customized memory hierarchy
- Performance/watt advantage over CPUs & GPUs

AWS F1 FPGA instance: Xilinx UltraScale+ VU9P

[Figure source: David Pellerin, AWS]
MEMORY CUSTOMIZATION
Case Study: Convolution

- The main computation of image/video processing is performed over overlapping stencils, termed as convolution.

\[
(I_{mg} \otimes f)_{\left[ n+i, m+j \right]}^{k-1,k-1} = \sum_{i=0}^{k-1} \sum_{j=0}^{k-1} I_{mg}[n+i][m+j] \cdot f[i,j]
\]
Example Application: Edge Detection

- Identifies discontinuities in an image where brightness (or image intensity) changes sharply
  - Very useful for feature extractions in computer vision

Sobel operator \( G=(G_X, G_Y) \)

\[
G_X = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}
\]

\[
G_Y = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}
\]

Figures: Pilho Kim, GaTech
CPU Implementation of a 3x3 Convolution

```
for (r = 1; r < R; r++)
    for (c = 1; c < C; c++)
        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];
```
General-Purpose Cache for Convolution

- Minimizes main memory accesses to improve performance

- A general-purpose cache is expensive in delay, area, and incurs nontrivial energy overhead
  - Nontrivial logic required for data access, placement, and replacement
Specializing Cache for Convolution

- Remove rows that are not in the neighborhood of the convolution window
Specializing Cache for Convolution

- Rearrange the rows as a 1D array of pixels
- Each time we move the window to right and push in the new pixel to the “cache”

Much simpler logic for data placement and replacement!
A Specialized “Cache”: Line Buffer

- Line buffer: a fixed-width “cache” with \((K-1)*C+K\) pixels in flight
  - Fixed addressing: Low area/power and high performance

- In customized FPGA implementation, line buffers can be efficiently implemented with on-chip BRAMs
DATA TYPE CUSTOMIZATION
Binary Number Representation

Unsigned number

- MSB has weight $2^{n-1}$
- Range of an n-bit unsigned number: ?

Two’s complement

- MSB has weight $-2^{n-1}$
- Range of an n-bit two’s complement number: ?

Examples: assuming integers here

<table>
<thead>
<tr>
<th>$2^3$</th>
<th>$2^2$</th>
<th>$2^1$</th>
<th>$2^0$</th>
<th>unsigned</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>= 11</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$-2^3$</th>
<th>$2^2$</th>
<th>$2^1$</th>
<th>$2^0$</th>
<th>2’c</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>= -5</td>
</tr>
</tbody>
</table>
Arbitrary Precision Integer

- C/C++ only provides a limited set of native integer types
  - `char` (8b), `short` (16b), `int` (32b), `long` (?), `long long` (64b)
  - Byte aligned: efficient in processors

- Arbitrary precision integer in Vivado HLS
  - Signed: `ap_int`; Unsigned `ap_uint`
  - Templatized class `ap_int<W>` or `ap_uint<W>`
    - `W` is the user-specified bitwidth
  - Two's complement representation for signed integer

```c
#include "ap_int.h"
...
ap_int<9> x; // 9-bit
ap_uint<24> y; // 24-bit unsigned
ap_uint<512> z; // 512-bit unsigned
```
Representing Fractional Numbers

- Binary representation can also represent fractional numbers, usually **called fixed-point numbers**, by simply extending the pattern to include negative exponents
  - Less convenient to use compared to floating-point types
  - Efficient and cheap in application-specific hardware

<table>
<thead>
<tr>
<th>2^3</th>
<th>2^2</th>
<th>2^1</th>
<th>2^0</th>
<th>2^-1</th>
<th>2^-2</th>
<th>unsigned</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>= 11.25</td>
</tr>
</tbody>
</table>

Binary point

```
1 0 1 1 0 1
```

```
-5 + 0.25 = -4.75
```
Overflow and Underflow

- **Overflow** occurs when a number is larger than the largest number that can be represented in a given number of bits.

<table>
<thead>
<tr>
<th>$2^5$</th>
<th>$2^4$</th>
<th>$2^3$</th>
<th>$2^2$</th>
<th>$2^1$</th>
<th>$2^0$</th>
<th>$2^{-1}$</th>
<th>$2^{-2}$</th>
<th><strong>unsigned</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>$= 11.25$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td>$= 11.25$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td>$= 11.25$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td>$= 3.25$</td>
</tr>
</tbody>
</table>

Drop MSB

Overflow occurs

- **Underflow** occurs when a number is smaller than the smallest number that can be represented.
Handling Overflow/Underflow

One common (& efficient) way of handling overflow / underflow is to drop the most significant bits (MSBs) of the original number, often called **wrapping**

<table>
<thead>
<tr>
<th>(-2^3)</th>
<th>2^2</th>
<th>2^1</th>
<th>2^0</th>
<th>2^-1</th>
<th>2^-2</th>
<th>2’c</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>(= -4.75)</td>
</tr>
</tbody>
</table>

Reduce integer width by 1
Wrap if overflows

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>(= ? )</td>
<td>(\approx 2.25)</td>
</tr>
</tbody>
</table>

Wrapping can cause a negative number to become positive, or a positive to negative
Fixed-Point Type in Vivado HLS

- Arbitrary precision fixed-point type
  - Signed: ap_fixed; Unsigned ap_ufixed
  - Templatized class ap_fixed<W, I, Q, O>
    - W: total word length
    - I: integer word length
    - Q: quantization mode
    - O: overflow mode
Example: Fixed-Point Modeling

- `ap_ufixed<11, 8, AP_TRN, AP_WRAP> x;`

```
MSB     LSB
b_7    ... b_1 b_0    ... b_3
```

- 11 is the total number of bits in the type
- 8 bits to the left of the decimal point
- AP_TRN defines truncation behavior for quantization
- AP_WRAP defines wrapping behavior for overflow
Fixed-Point Type: Overflow Behavior

- **ap_fixed overflow mode**
  - Determines the behavior of the fixed point type when the result of an operation generates more precision in the **MSBs** than is available

```
ap_fixed<W, IW_X> x;
ap_fixed<W, IW_Y> y = x; /* IW_Y < IW_X */
```

Default: **AP_WRAP** (wrapping mode)

AP_SAT (saturation mode)
Fixed-Point Type: Quantization Behavior

- **ap_fixed quantization mode**
  - Determines the behavior of the fixed point type when the result of an operation generates more precision in the **LSBs** than is available
  - Default mode: AP_TRN (truncation)
  - Other rounding modes: AP_RND, AP_RND_ZERO, AP_RND_INF, ...

```plaintext
ap_fixed<4, 2, AP_TRN>  x = 1.25;  (b’01.01)
ap_fixed<3, 2, AP_TRN>  y = x;    1.0  (b’01.0)

ap_fixed<4, 2, AP_TRN>  x = -1.25; (b’10.11)
ap_fixed<3, 2, AP_TRN>  y = x;    -1.5 (b’10.1)
```
E-D-A Revisited

- **Exponential**
  - in complexity (or Extreme scale)

- **Diverse**
  - increasing system heterogeneity
  - multi-disciplinary

- **Algorithmic**
  - intrinsically computational
Analysis of Algorithms

- Need a systematic way to compare two algorithms
  - Runtime is often the most common criterion used
  - Space (memory) usage is also important in most cases
  - But difficult to compare in practice since algorithms may be implemented in different machines, use different languages, etc.
  - Additionally, runtime is usually input-dependent.

- **big-O** notation is widely used for asymptotic analysis
  - Complexity is represented with respect to some natural & abstract measure of the problem size $n$
Big-O Notation

- Express runtime as a function of input size $n$
  - Runtime $F(n)$ is of order $G(n)$, written as $F(n) = O(G(n))$ when
    - $\exists n_0, \forall n \geq n_0, F(n) \leq KG(n)$ for some constant $K$
  - $F$ will not grow larger than $G$ by more than a constant factor
  - $G$ is often called an “upper bound” for $F$

- Interested in the worst-case input & the growth rate for large input size
Big-O Notation (cont.)

How to determine the order of a function?
- Ignore lower order terms
- Ignore multiplicative constants

Examples:
- $3n^2 + 6n + 2.7$ is $O(n^2)$
- $n^{1.1} + 10000000000n$ is $O(n^{1.1})$, $n^{1.1}$ is also $O(n^2)$

- $n! > C^n > n^c > \log n > \log \log n > C$
  \[ \Rightarrow n > \log n, \ n \log n > n, \ n! > n^{10}. \]

What do asymptotic notations mean in practice?
- If algorithm A is $O(n^2)$ and algorithm B is $O(n \log n)$, we usually say algorithm B is more scalable.
Asymptotic Notions

- **big-Omega** notation $F(n) = \Omega(G(n))$
  - $\exists n_0, \forall n \geq n_0, F(n) \geq Kg(n)$ for some constant $K$
  - $G$ is called a “lower bound” for $F$

- **big-Theta** notation $F(n) = \Theta(G(n))$
  - if $G$ is both an upper and lower bound for $F$
  - Describes the growth of a function more accurately than $O(\ldots)$ or $\Omega(\ldots)$
  - Examples:
    - $4n^2 + 1024 = \Theta(n^2)$
    - $n^3 + 4n \neq \Theta(n^2)$
Exponential Growth

- Consider $2^n$, value doubled when $n$ is increased by 1

<table>
<thead>
<tr>
<th>$n$</th>
<th>$2^n$</th>
<th>1 ns (/op) x $2^n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>$10^3$</td>
<td>1 us</td>
</tr>
<tr>
<td>20</td>
<td>$10^6$</td>
<td>1 ms</td>
</tr>
<tr>
<td>30</td>
<td>$10^9$</td>
<td>1 s</td>
</tr>
<tr>
<td>40</td>
<td>$10^{12}$</td>
<td>16.7 mins</td>
</tr>
<tr>
<td>50</td>
<td>$10^{15}$</td>
<td>11.6 years</td>
</tr>
<tr>
<td>60</td>
<td>$10^{18}$</td>
<td>31.7 years</td>
</tr>
<tr>
<td>70</td>
<td>$10^{21}$</td>
<td>31710 years</td>
</tr>
</tbody>
</table>
NP-Complete

- The class **NP-complete** (NPC) is the set of decision problems which we “believe” there is no polynomial time algorithms (hardest problem in NP)

- **NP-hard** is another class of problems, which are at least as hard as the problems in NPC (also containing NPC)

- If we know a problem is in NPC or NP-hard, there is (very) little hope to solve it exactly in an efficient way
How to Identify an NP-Complete Problem

- I can’t find an efficient algorithm, I guess I’m just too dumb.

- I can’t find an efficient algorithm, because no such algorithm is possible.

- I can’t find an efficient algorithm, but neither can all these famous people.

[source: Computers and Intractibility by Garey and Johnson]
Reduction

- Showing a problem P is not easier than a problem Q
  - Formal steps:
    - Given an instance q of problem Q,
    - there is a polynomial-time transformation to an instance p of P
    - q is a “yes” instance iff p is a “yes” instance
  - Informally, if P can be solved efficiently, we can solve Q efficiently (Q is reduced to P)
    - P is polynomial time solvable $\rightarrow$ Q is polynomial time solvable
    - Q is not polynomial time solvable $\rightarrow$ P is not polynomial time solvable

- Example:
  - Problem A: Sort $n$ numbers
  - Problem B: Given $n$ numbers, find the median
Problem Intractability

- Most of the nontrivial EDA problems are intractable (NP-complete or NP-hard)
  - Best-known algorithm complexities that grow exponentially with n, e.g., \( O(n!) \), \( O(n^n) \), and \( O(2^n) \).
  - No known algorithms can ensure, in a time-efficient manner, globally optimal solution

- **Heuristic** algorithms are used to find near-optimal solutions
  - Be content with a “reasonably good” solution
Many Algorithm Design Techniques

- There can be many different algorithms to solve the same problem
  - Exhaustive search
  - Divide and conquer
  - Greedy
  - Dynamic programming
  - Network flow
  - ILP
  - Simulated annealing
  - Evolutionary algorithms
  - ...
Types of Algorithms

- There are many ways to categorize different types of algorithms
  - Polynomial vs. Exponential, in terms of computational effort
  - Optimal (exact) vs. Heuristic, in terms of solution quality
  - Deterministic vs. Stochastic, in terms of decision making
  - Constructive vs. Iterative, in terms of structure
  ...

...
Broader List of Algorithms

- Combinatorial algorithms
  - Graph algorithms
  ...

- Computational mathematics
  - Optimization algorithms
  - Numerical algorithms
  ...

- Computational science
  - Bioinformatics
  - Linguistics
  - Statistics
  ...

- Information theory & signal processing

- Many more ...

Next Class

- Graph algorithms
  - Timing analysis
  - BDDs
Acknowledgements

- These slides contain/adapt materials from / developed by
  - Prof. David Pan (UT Austin)
  - “VLSI Physical Design: From Graph Partitioning to Timing Closure” authored by Prof. Andrew B. Kahng, Prof. Jens Lienig, Prof. Igor L. Markov, Dr. Jin Hu