# CS220A Lab#6
# Handling Large Inputs

Mainak Chaudhuri
Indian Institute of Technology Kanpur

# Sketch

- Assignment#1: 7-bit adder/subtractor
- Assignment#2: walk in a square grid

# Lab#6

- Make new folders Lab6_1, Lab6_2 under CS220Labs to do the assignments
- Refer to lab#1 slides for Xilinx ISE instructions
- Finish pending assignments from lab#5 first
- Marks
  - Assignment#1: 3 marks
  - Assignment#2: 5 marks

# Assignment#1

- 7-bit adder/subtractor
  - In this assignment, you will design a 7-bit adder/subtractor that takes two 7-bit numbers A and B in two's complement representation and a one-bit operation code (add=0, sub=1) and produces a 7-bit two's complement result and an overflow bit
  - The input A will be taken in two rotation steps of the shaft encoder with the help of the slide switches (lower 4 bits first and the next 3 bits)
    - Place the switches in position and rotate the shaft one step, detect the rotation event and read in the input bits from the switches

# Assignment#1

- 7-bit adder/subtractor
  - After input A is accepted, input B will be taken similarly in two rotation steps of the shaft
  - After input B is accepted, the operation code will be taken from slide switch SW0 in one rotation step of the shaft encoder
  - Internally, you have to count the number of rotation steps to figure out which portion of which input you are currently accepting
  - The 7-bit output result will be displayed in LED0 to LED6 and LED7 should glow only if there is an overflow
    - Carry out is not shown

# Assignment#1

- 7-bit adder/subtractor
  - Verilog modules
    - A module to do one-bit addition/subtraction
      - Four inputs: a, b, cin, add/sub; Two outputs: sum, cout
    - Use the module from the last lab to detect rotation events
    - Change the module from the last lab that detects rising edge of rotation events; on a rising edge of rotation event, read the inputs; outside the always block, instantiate an array of seven adder/subtractors and compute the overflow
    - Write a module that instantiates the aforementioned last two modules and connects them
  - Use PlanAhead to assign pins and synthesize the top-level module (No need to simulate in ISim)

# Assignment#2

- Walk in a square grid
  - Imagine a 15x15 grid (length of each side is 15)
    - The leftmost bottom corner is (0, 0)
    - The leftmost top corner is (0, 15)
    - The rightmost top corner is (15, 15)
    - The rightmost bottom corner is (15, 0)
  - A worm is sitting at (0, 0) to start with
  - At each move, the worm can take 0, 1, 2, or 3 steps along east, west, north, or south directions
    - If the move causes the worm to hit the boundary, it stops there
    - For example, at (0, 0) if it tries to move toward west or south, it stays at (0, 0); at (13, 0) if it tries to move toward east three steps, it takes only two steps and stops at (15, 0)

# Assignment#2

- Walk in a square grid
  - In this assignment, at each move, the inputs are the number of steps (2 bits) and the direction of the move (2 bits)
  - Use the slide switches to take the four bits of inputs
  - A move of the worm is defined by one step rotation of the rotary shaft encoder
    - Place the slide switches in position, rotate the shaft, and read the inputs
  - After accepting the inputs at each move, compute the new coordinates of the worm using a five-bit adder/subtractor
    - Sign bit is always zero because the operands are always positive

# Assignment#2

- Walk in a square grid
  - Display the new coordinates in the LEDs
    - Use LED0, LED1, LED2, LED3 for x-coordinate and the remaining four for the y-coordinate
  - Plan the Verilog modules
    - You can reuse a lot from the previous assignment
  - Use PlanAhead for pin assignment and synthesize the hardware
  - No need to simulate in ISim
  - Hint: use the MSB of the five-bit adder/subtractor's sum output to check if the worm has hit the boundary
    - This is not exactly an overflow because any negative result or a positive result more than 15 would mean that the worm has hit the boundary