# ECE 3401 Digital Systems Design – Spring 2025

## Programming Assignment 1:
## Datapath Logic for LCM and GCD Digital Design

*Due February 17, 2025 (Monday) @ 11:59 PM on HuskyCT*

This programming assignment implements the digital design of parallel datapath components for least common multiple (LCM) and greatest common divisor (GCD) computations for two numbers. There are multiple ways to compute LCM and GCD. Figures 1 and 2 include the algorithmic implementation for this assignment. For LCM, the code initializes two variables “a” and “b” with the values of inputs “x” and “y”. A loop runs until “a” equals “b”, indicating the LCM is found. If “a” is smaller than “b”, “x” is added to “a”; otherwise, “y” is added to “b”. When “a” == “b”, it returns the LCM. GCD algorithm starts by assigning the input values “x” and “y” to “a” and “b”. A loop runs until “a” equals “b”, indicating the GCD is found. In each iteration, if “a” is smaller than “b”, the algorithm subtracts “a” from “b”; otherwise, it subtracts “b” from “a”. Finally, it returns the GCD when both numbers become equal.

```python
def compute_lcm(x, y):
    a = x
    b = y
    print("a", a, "b", b)
    cycles = 0

    while(a != b):
        cycles = cycles + 1
        if(a < b):
            a = a + x
            b = b
            print("a", a, "b", b)
        else:
            a = a
            b = b + y
            print("a", a, "b", b)
    print("cycles: ", cycles)
    return a
```

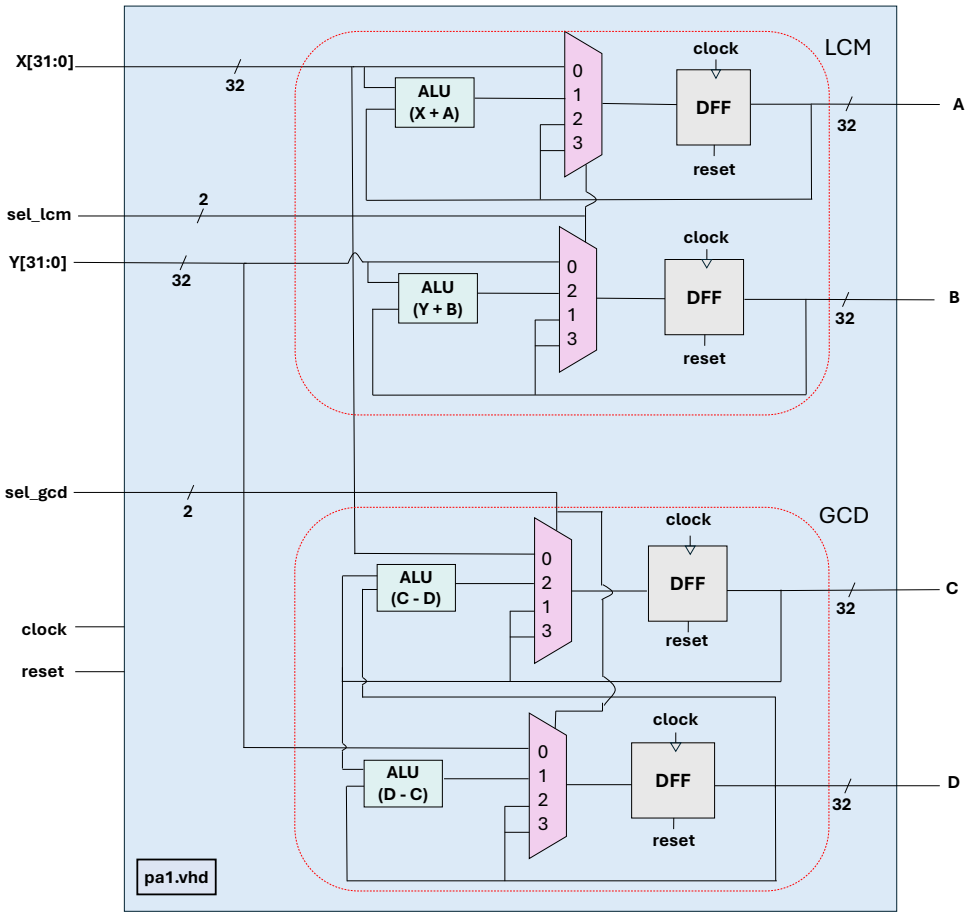
*Figure 1: Python code for LCM*

```python
def compute_gcd(x, y):
    a = x
    b = y
    print("a", a, "b", b)
    cycles = 0

    while(a != b):
        cycles = cycles + 1
        if(a < b):
            a = a
            b = b - a
            print("a", a, "b", b)
        else:
            a = a - b
            b = b
            print("a", a, "b", b)
    print("cycles: ", cycles)
    return a
```

*Figure 2: Python code for GCF*

We will implement a digital design capable of computing both LCM and GCD of two numbers in parallel. The design receives 32-bit input operands, as well as separate 2-bit select signals to independently control the datapath for computing LCM and GCD. Moreover, the design receives the clock and reset signals as inputs. The datapath for the design is shown in the figure below. The LCM unit is implemented with two 32-bit ALUs, two 32-bit D flip-flop (DFF) banks and two 4:1 multiplexors for control. The GCD unit uses explicit datapath components (two ALUs, two D flip flop banks and two multiplexors) for its computations.




You will enter the VHDL designs for the ALU and DFF modules and port map four instances of each in the pa1.vhd. The 2-bit select signals are used to control the four 4:1 multiplexors, and they are derived using the computed four 32-bit outputs, A, B, C and D.

For LCM, following Figure 1, when the design starts after reset, it must capture the inputs (X and Y) using sel_lcm=0. During LCM iterations, when A < B, sel_lcm =1 to hold B's DFF output and capture the output of ALU (that is configured to perform X + A) in A's DFF. In case, A > B, sel_lcm =2 to hold A's DFF output and capture the output of ALU (configured to perform Y + B) in B's DFF. When LCM algorithm completes (i.e., A == B), sel_lcm=3 to hold A and B DFFs to ensure the LCM outcome is available (DFF A hold the LCM output) until the design is reset for next LCM computations.

Concurrent to the LCM, the GCD following Figure 2 starts after reset by capturing the inputs (X and Y) using sel_gcd=0. During GCD iterations, when C < D, sel_gcd =1 to hold C's DFF output and capture the output of ALU (that is configured to perform D - C) in D's DFF. In case, C > D, sel_gcd =2 to hold D's DFF output and capture the output of ALU (configured to perform C - D) in C's DFF. When GCD algorithm completes (i.e., C == D), sel_gcd=3 to hold C and D DFFs to ensure the GCD outcome is available (DFF C hold the GCD output) until the design is reset for next GCD computations.

You must use concurrent statements in pa1.vhd to implement the four multiplexors.

Each arithmetic logical unit (ALU) can either perform 32-bit addition or subtraction. We do not require structural VHDL for these operations. You should use behavioral operators and the appropriate IEEE library (IEEE.numeric_bit.all library) for your design.

The DFF is implemented as 32-bit D flip-flop-based registers with an asynchronous reset i.e., anytime the reset is set to "1", the DFF must reset to 0s regardless of the clock signal. When reset is "0", and on the rising edge of the clock, the registers capture the input. Otherwise, the registers hold their values.

At the time of a new LCD and/or GCD computation, the register output values should be '0' by asserting the reset signal.

You will use separate VHDL modules for the register bank (dff.vhd), the ALU (alu.vhd), and the overall PA1 module (pa1.vhd). You are given top-level modules, and you are expected to write the architecture for each module. You will be graded on the design of these modules and their functionality.

## DESIGN VERIFICATION

You are expected to test your design using a series of testbenches. We have provided a fully functional testbench1.vhd that only computes the LCM if 7 and 5. It follows this structure:

- Reset by asserting reset="1" and wait for a clock period
- Clear the reset signal. sel_lcm is set to 0 to load the value of X (7) and Y (5) into registers A and B. Wait for a clock period
- Execute a while loop until A /= B. After each iteration wait for a clock period
    - When A < B, set sel_lcm to "01", else when A > B, sel_lcm to "10", else sel_lcm to "11"
    - Increment the "cycles_lcm" on each iteration of loop
- When A ==B, lcm signal is set to A. Wait for a clock period. Now the LCM output can be readout.

Your pa1.vhd must be completed before testing it with testbench1.vhd. Further, you are expected to design three testbenches for this assignment.

**Testbench2.vhd: Computing GCD only for numbers 7 and 5**

- Reset by asserting reset="1" and wait for a clock period
- Clear the reset signal. sel_gcd is set to 0 to load the value of X (7) and Y (5) into registers C and D. Wait for a clock period
- Execute a while loop until C /= D. After each iteration wait for a clock period
    - When C < D, set sel_gcd to "01", else when C > D, sel_gcd to "10", else sel_gcd to "11"
    - Increment the "cycles_gcd" on each iteration of loop
- When C == D, gcd signal is set to C. Wait for a clock period. Now the GCD output can be readout.

**Testbench3.vhd: Computing LCM & GCD in parallel for numbers 7 and 5**

- Reset by asserting reset=”1” and wait for a clock period
- Clear the reset signal. sel_lcm and sel_gcd are set to 0 to load the value of X (7) and Y (5) into registers A, B, C and D. Wait for a clock period
- Perform both the LCM and GCD operation in parallel and keep the loop running until both the lcm and gcd has been computed. Note that LCM or GCD may complete at different cycles, so the select signals must hold the computed values. The cycles counters must also only account for the cycles when a LCM or GCD is being computed.
- After the computation is done, set the lcm and gcd signal to the computed LCM and GCD values in DFF A and C respectively. Wait for a clock period

**Testbench4.vhd:**

- Reset the system and wait for a cycle
- Clear reset signal, set the input X ← 7 and Y ← 5 and sel_lcm=”00”. Wait for a cycle
- Perform only the LCM computation while waiting a cycle for each iteration.
- When LCM is done, set lcm signal and assert reset for one cycle
- Clear reset signal, set the input X ← 7 and Y ← 5 and sel_gcd=”00”. Wait for a cycle
- Perform only the GCD computation while waiting a cycle for each iteration.
- When GCD is done, set gcd signal and assert reset for one cycle
- Repeat the previous steps for new inputs X ← 54 and Y ← 24
- Now compute parallel LCM and GCD for inputs X ← 7 and Y ← 5 using the template from testbench3

**DELIVERABLES**

Please submit the following report saved as a single PDF:

1. Your code for each module and your testbenches. You can copy and paste the code into a Word document; make sure to clearly label each code block.
   alu.vhd, dff.vhd, pa1.vhd, testbench2.vhd, testbench3.vhd, and testbench4.vhd

2. Submit screenshots of the following: Your output waveform of testbench1.vhd, testbench2.vhd and testbench3.vhd from 0ns to 300ns whereas for testbench4.vhd, output waveform should be from 0ns to 600ns . The clk, reset, X, Y, A, B, C, D, sel_lcm, sel_gcd, cycles_lcm, cycles_gcd, lcm and gcd should all be clearly visible. The data format should also be unsigned decimal radix for all of these signals. Format all the waveforms as described.