# COL216 Lab Assignment - 0

## Variable Latency Multiplier

Yash Bansal 2022CS511333
Shivam Sawarn 2022CS11075

## 1 Problem Statement

1. Implement a fixed-latency fixed-point multiplier in VHDL to multiply two 16-bit unsigned binary numbers.
2. Enhance the multiplier using Booth's multiplication algorithm to create a variable-latency multiplier. Compare the performance of the two algorithms for multiple random inputs.

## 2 Design Principles

### 2.1 Adder

Initially, a 16-bit adder circuit was designed to serve as a component in the multiplier circuit. The design incorporates principles from both carry-lookahead adder and ripple carry adder methodologies:

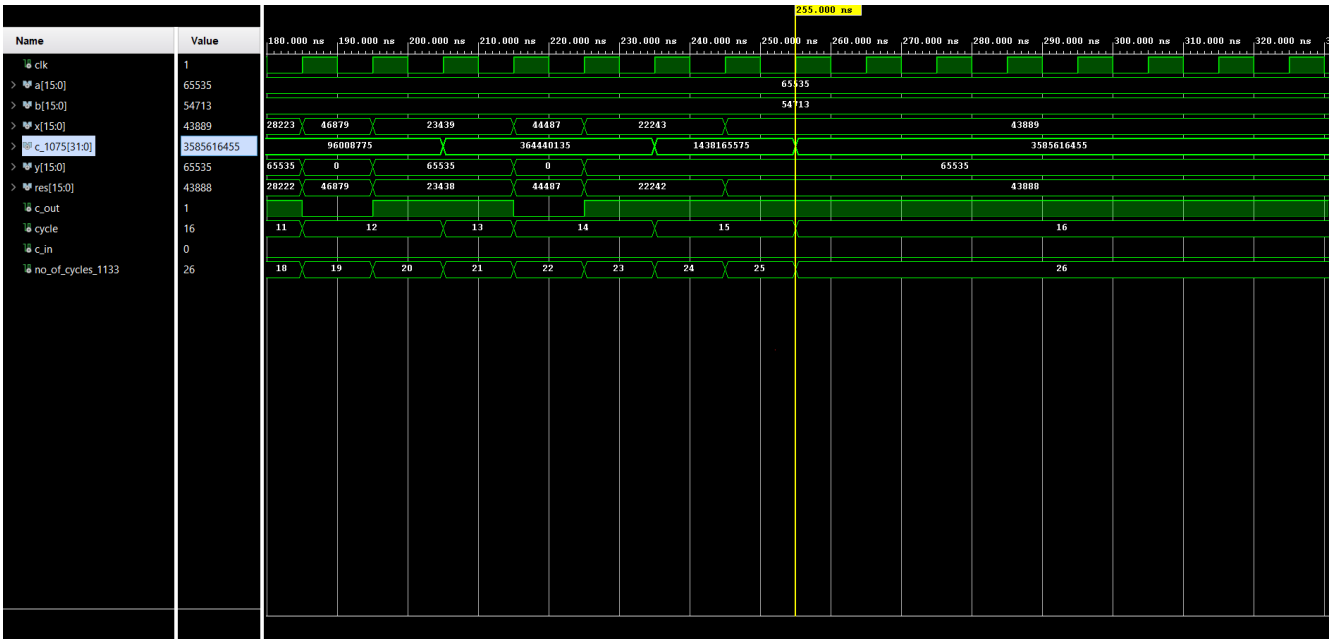
- Four full-adder circuits, each of 4 bits, are designed. A carry-lookahead adder is employed to maintain the carry, with the initial carry of the 1st adder set to 0.
- Following the ripple carry adder principle, the carry of the 1st adder ripples to the 2nd adder and so on until the outputs of all four adders are computed.
- The final carry, denoted as `c_out`, is stored as part of the output signal.

### 2.2 Fixed-point Multiplier

In the fixed-point multiplier, the following steps are done until our variable cycle is 16:

- The variable cycle represents the position of the multiplier bit and the corresponding position of the addition in the result. The result is initialized as a 32-bit vector (`c_1075`) with all bits set to '0'.
- If the current multiplier bit is '0', the algorithm moves to the next bit without computation, taking one clock cycle.
- For the case when the multiplier bit is '1', an additional variable `cycle2` is introduced. When `cycle2` is '0', a required 16-bit number is extracted from `c_1075` and added to the multiplicand. After this, `cycle2` is set to '1'.
- When `cycle2` is '1', the result of the addition (computed from `cycle2 = '0'`) is assigned back to `c_1075` at the appropriate position, making it a two-clock cycle operation for cases where the multiplier bit is '1'.

- When the variable cycle reaches 16, the final result is computed and stored in the vector `c_1075`.




a(multiplicand) = 65535 b(multiplier) = 54713

c_1075(final result) = 3585616455 no_of_cycles_1133 = 26

## 2.3 Subtracter

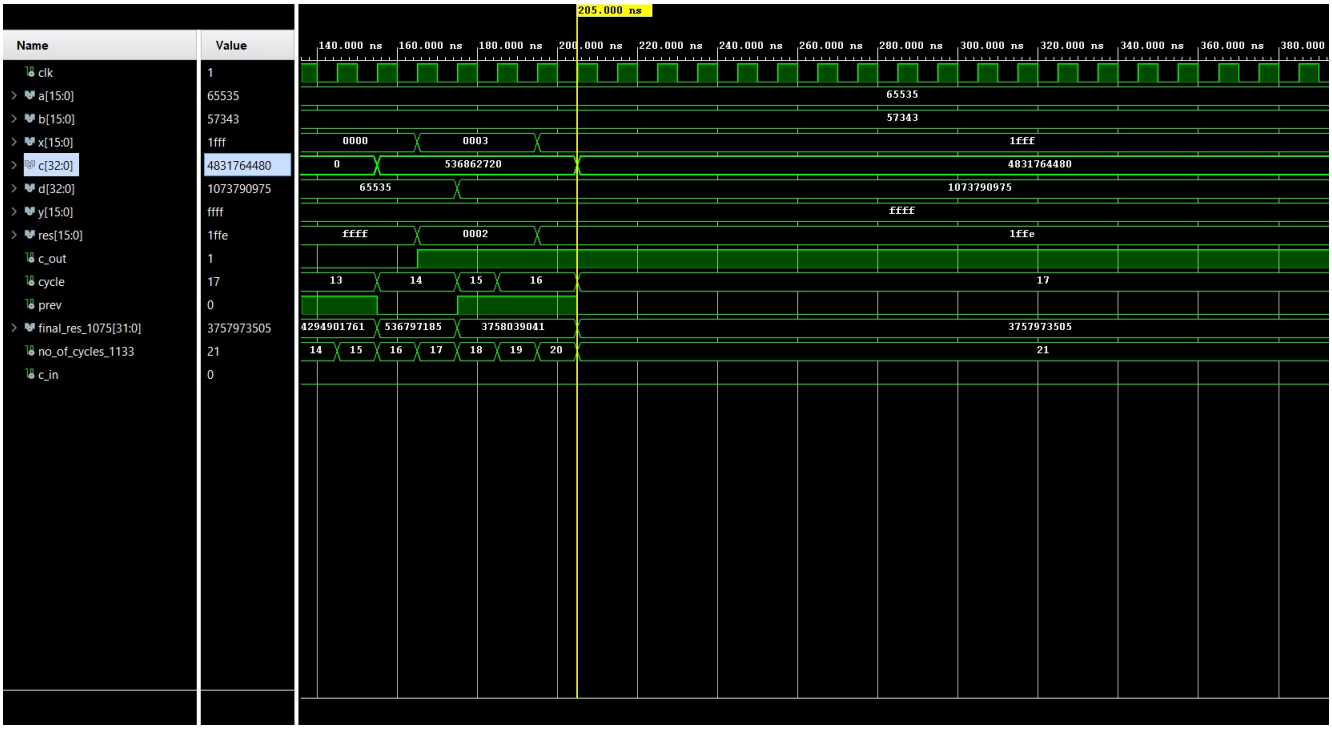
A 32-bit subtracter is designed to be utilized as a component for Booth's multiplication algorithm:

- Similar to the adder circuit, a 4-bit carry-lookahead adder and ripple carry adders are combined, with the initial carry set to '1'.
- Before the subtraction operations, the bits of the second number are inverted to represent its 2's complement form.

## 2.4 Booth's Multiplier

In Booth's multiplier, the following steps are performed for a variable cycle count of 17:

- Two variables, `c` and `d`, each of 32 bits, are created and initialized to 0. The final answer is stored in `final_res_1075`.
- When the current multiplier bit is '1' and the previous bit is '0', an additional variable `cycle2` is used. When `cycle2` is zero, a required 16-bit number is extracted from the vector `d` and added to the multiplicand. After this, `cycle2` is assigned '1'. When `cycle2` is '1', the result of the addition is assigned back to `d` at the appropriate position. The vector `d` represents the sum of all the numbers subtracted in Booth's algorithm.
- When the current multiplier bit is '0' and the previous bit is '1', a similar process is followed with `c` representing the sum of all the numbers added in Booth's algorithm.
- The final result is obtained by subtracting `c` and `d`, and the result is stored in `final_res_1075`.




a(multiplicand) = 65535

b(multiplier) = 57343

c(booth_addition) = 4831764480

d(booth_subtraction) = 1073790975

final_res_1075(final result) = 3757973505

no_of_cycles_1133 = 21

# 3 Note:-

- main2.vhd is implementation of fixed-point multiplication and main3.vhd is the implementation of booth multiplier.