Booth Multiplier

Booth's Multiplication Algorithm is an efficient method for multiplying two signed binary numbers in 2's complement notation. It reduces the number of addition and subtraction operations by encoding the multiplier. This documentation provides an overview of the Verilog implementation of Booth's Algorithm using a Finite State Machine (FSM).

1. Registers and Variables Used

- **A** (Accumulator) - Stores intermediate results during the multiplication process.
- **M** (Multiplicand) - The number to be multiplied.
- **Q** (Multiplier) - The number by which M is multiplied.
- **Qres** (Residual Bit) - Stores the previous least significant bit (Q0) for decision making.
- **n** (Counter) - Tracks the number of iterations (equal to the bit-width of Q).

2. Algorithm Steps

Step 1 : Initialization

- Load initial values:
    - A = 0
    - Qres = 0
    - M = Multiplicand
    - Q = Multiplier
    - n = Bit-width of Q (number of iterations)

Step 2 : Checking Q and Qres

- If **Q0, Qres = 00** or **Q0, Qres = 11**, go to Step 5.
- If **Q0, Qres = 01**, go to Step 3.
- If **Q0, Qres = 10**, go to Step 4.

Addition/Subtraction Steps

- **Step 3**: Perform $A = A + M$, then go to Step 5.
- **Step 4**: Perform $A = A - M$, then go to Step 5.
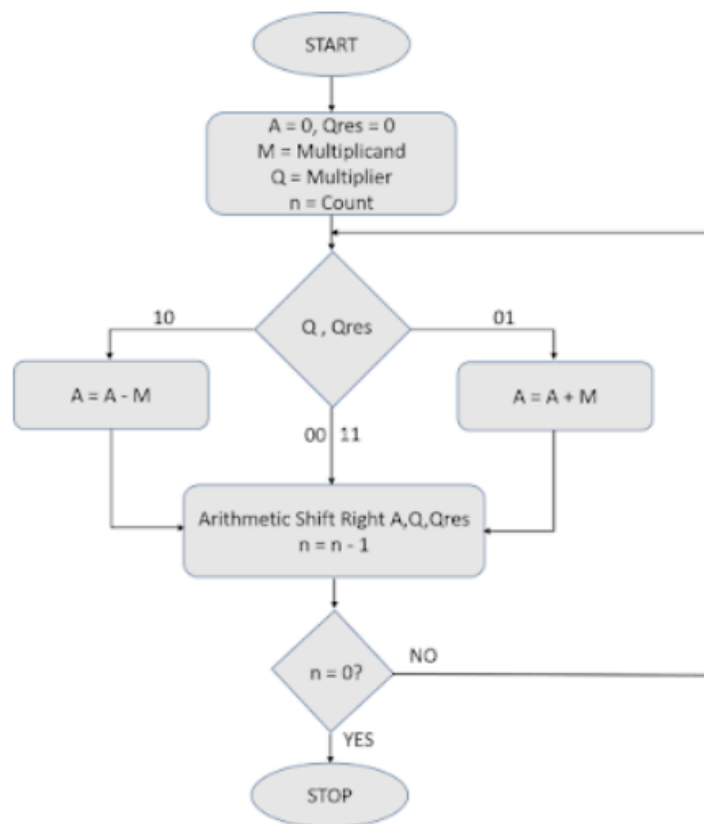
Step 5 : Arithmetic Shift Right

- Perform an arithmetic right shift on {A, Q, Qres}.
- Decrement n.

Step 6 : Check Counter

- If n == 0, stop execution.
- Else, go back to Step 2.

Step 7 : Result Extraction

- The final product is stored in {A, Q}.

Verilog Code Logic:

Here, we are going to implement the Booth Multiplication algorithm for 3-bit operands (1 sign bit) in an FSM format with completely synthesizable Verilog Code.

Inputs:

clk, rst, X, Y (2 operands to be multiplied), start (This pulse indicates start of computation)

Outputs:

Z (Product) and valid (This pulse indicates the availability of final product)

1. In IDLE state, we wait for the start pulse. Upon its arrival, move to START state.
2. In START state, we follow the same algorithm and perform the computations using verilog logic as long as counter < 3. (Incrementing counter is used)
3. Upon counter completion, we send out valid pulse and goto IDLE state.