**Booth’s Multiplier**

­­

**Revision 1.0**

**Table of Contents**

[**Introduction** 4](#_Toc151455328)

[**What is Booth’s Multiplier** 5](#_Toc151455329)

[**Booth’s Algorithm Implementation Architecture** 6](#_Toc151455330)

[**Top Module** 6](#_Toc151455331)

[**Signal Description** 6](#_Toc151455332)

[**Input Signals** 6](#_Toc151455333)

[**Output Signals** 6](#_Toc151455334)

[**Data Path and Control Path** 7](#_Toc151455335)

[**Control Path** 8](#_Toc151455336)

[**Signal Description** 8](#_Toc151455337)

[**Input Signals** 8](#_Toc151455338)

[**Output Signals** 8](#_Toc151455339)

[**Control Path FSM** 9](#_Toc151455340)

[**Control Path Algorithm** 9](#_Toc151455341)

[**Datapath** 10](#_Toc151455342)

[**Input Signals** 10](#_Toc151455343)

[**Output Signals** 11](#_Toc151455344)

**List of Illustrations**

**Figures**

[Figure 1. top\_booths\_multiplier 7](#_Toc151460773)

[Figure 2. Data Path and Control Path 7](#_Toc151460774)

[Figure 3. Control path Moore FSM 9](#_Toc151460775)

[Figure 4. Booths’ Multiplier Algorithm 10](#_Toc151460776)

[Figure 5. DataPath 10](#_Toc151460777)

# **Introduction**

Booth's algorithm, also known as Booth's multiplication algorithm, is a computer algorithm used for binary multiplication. It was developed by Andrew Donald Booth in 1951 and is particularly useful for efficiently multiplying two binary numbers, especially when one of the numbers is negative or when signed binary representation is involved. Booth's algorithm reduces the number of required additions and subtractions during the multiplication process, making it more efficient than traditional methods.

Using other multiplication techniques requires chain of half-adders and numerous gates. Due to which they increase the combinational delay of the system.

As this algorithm follows certain principles for multiplying two signed or unsigned numbers. It becomes comparatively easy to minimize the overall logic of the system with minimal gate delays.

# **What is Booth’s Multiplier**

# **Booth’s Algorithm Implementation Architecture**

## **Top Module**

The top module consists of four input signals and two output signals. Being a top module, rest of the blocks are enclosed inside the tb\_top module.

## **Signal Description**

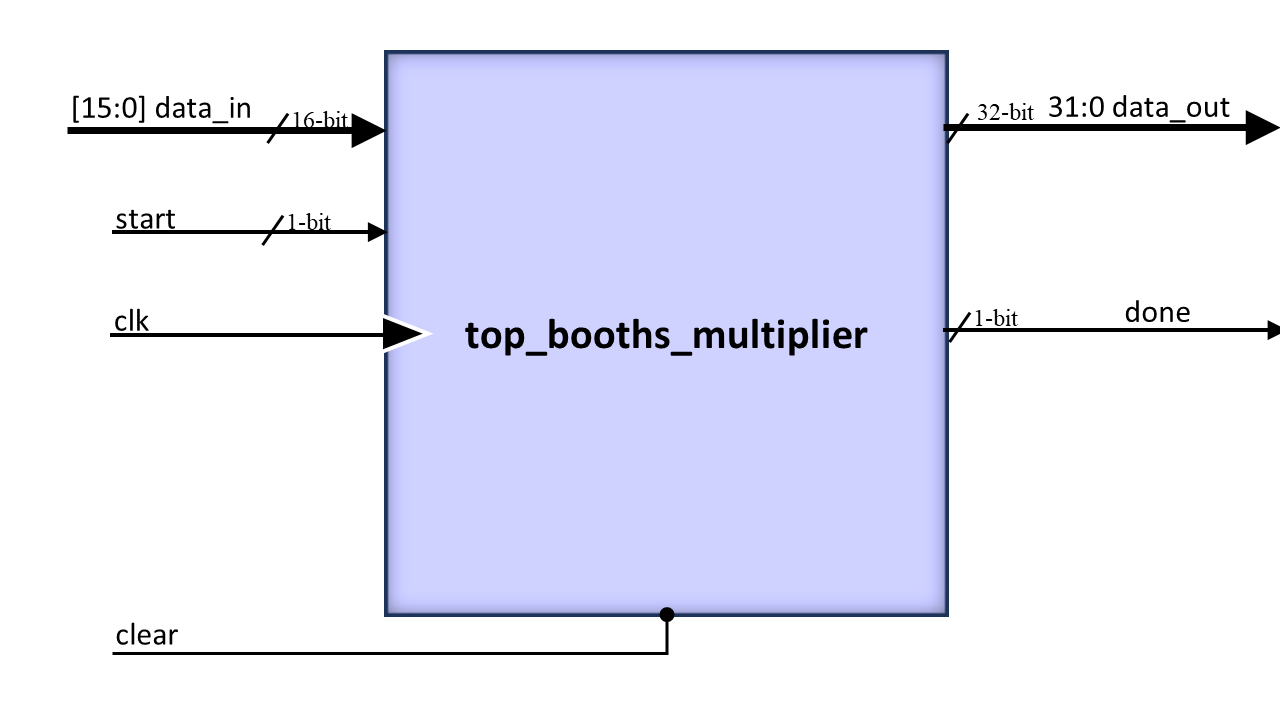
### **Input Signals**

* 1. **clear:** System is sensitive to active-low asynchronous clear.
  2. **clk:**  It is pulsating synchronizing signal having 50 percent duty cycle. The system is positive clock edge sensitive.
  3. **start:** When HIGH, activates the control path.
  4. **[15:0] data\_in:** 16-bit input signal. The multiplier and multiplicand are fed to the data path by toggling respective control signals via control path.

### **Output Signals**

**1. done:** Flag signal which indicates correct result is ready at the data\_out pin.

**2. [31:0 ] data\_out:** Resultant 32-bit output as input is of 16-bit.

  
Figure 1. top\_booths\_multiplier

## **Data Path and Control Path**

Whole design is segregated into datapath and controlpath. The control path generates control signals and datapath performs data operations as per the control signals.

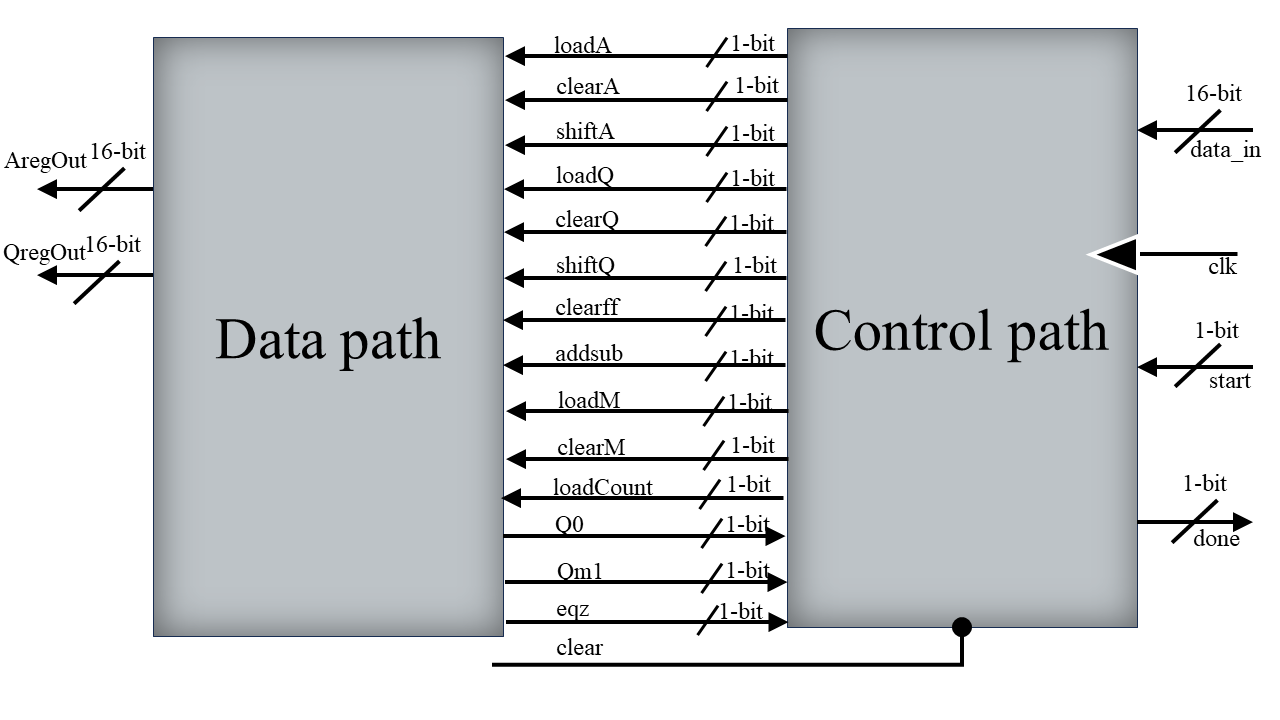


Figure 2. Data Path and Control Path

## **Control Path**

The control unit consists of control signals which are fed to the datapath. It is FSM (Finite State Machine) based control path which transits from one state to another in Moore machine style.

### **Signal Description**

### **Input Signals**

1. **clear:**  control unit is sensitive to active log asynchronous clear.

2. **clk:** It is pulsating synchronizing signal having 50 percent duty cycle. The system is positive clock edge sensitive.

3. **clear:** Active-Low asynchronous clear signal.

4. **start:** Initiates the working of controlpath. It is an active-high signal.

5. **eqz:** When HIGH, indicates counter has completed counting.

6 **Q0:** LSB of multiplicand. Intermediate 1-bit results are taken as input for state transition.

7.**Qm1:** Intermediate 1-bit result taken as input for state transition.

### **Output Signals**

1. **loadA:** When HIGH, loads the A register in datapath with ALU output.

2. **clearA:** When HIGH, clears the register-A in datapath.

3. **shiftA:** When HIGH, shifts the content of A register in datapath by 1-bit.

4. **loadQ:** When HIGH, loads the Q register in datapath with register-A output.

5. **clearQ:** When HIGH, clears the register-Q in datapath.

6. **shiftQ:** When HIGH, shifts the content of register-Q in datapath by 1-bit.

7. **loadM:** When HIGH, loads the register-A in datapath with data\_in.

8. **clearM:** When HIGH, clears the register-A in datapath with data\_in.

9. **clearff:** When HIGH, clears the Qm1 register

10. **addsub:** When HIGH, performs addition of contents of register-A and register-M. When, LOW, performs subtraction of contents of register-A and register-M.

11. **decr:** When HIGH, signals the counter to decrement

12. **loadCount:** When HIGH, signals the counter to start counting.

13. **done:** When HIGH, resembles correct output is present at the output lines of registers.

## **Control Path FSM**

The control path as discussed generates output signals as per the value of Qm1 and Q0. Below is the state transition diagram of Moore machine used in the control path design. To emphasize more on the table, it is followed by a pictorial representation of the booths’ algorithm multiplier.

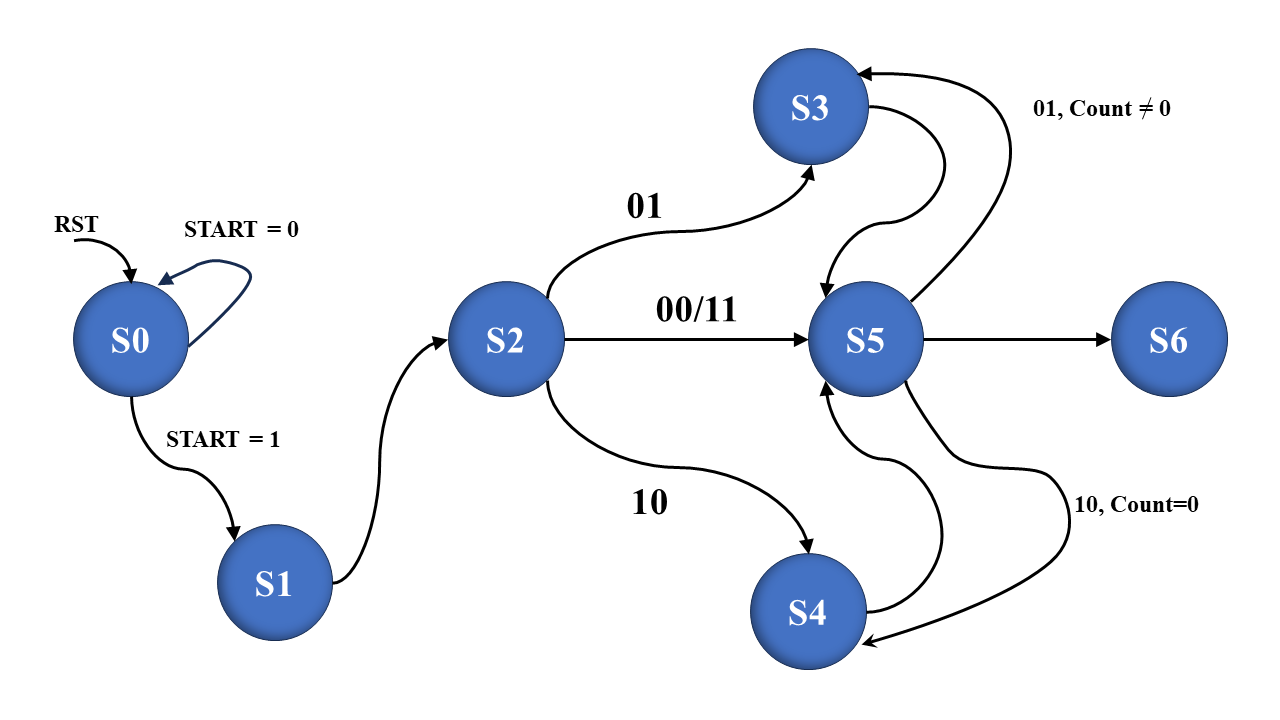


Figure 3. Control path Moore FSM

## **Control Path Algorithm**

As booths’ algorithm multiplier heavily relies on shifting and addition or subtraction, therefore it is necessary to toggle appropriate signals at certain stages. FSM ensures this behaviour so that error is minimized.

The algorithm is sensitive to the content of Qm1 and Q0.

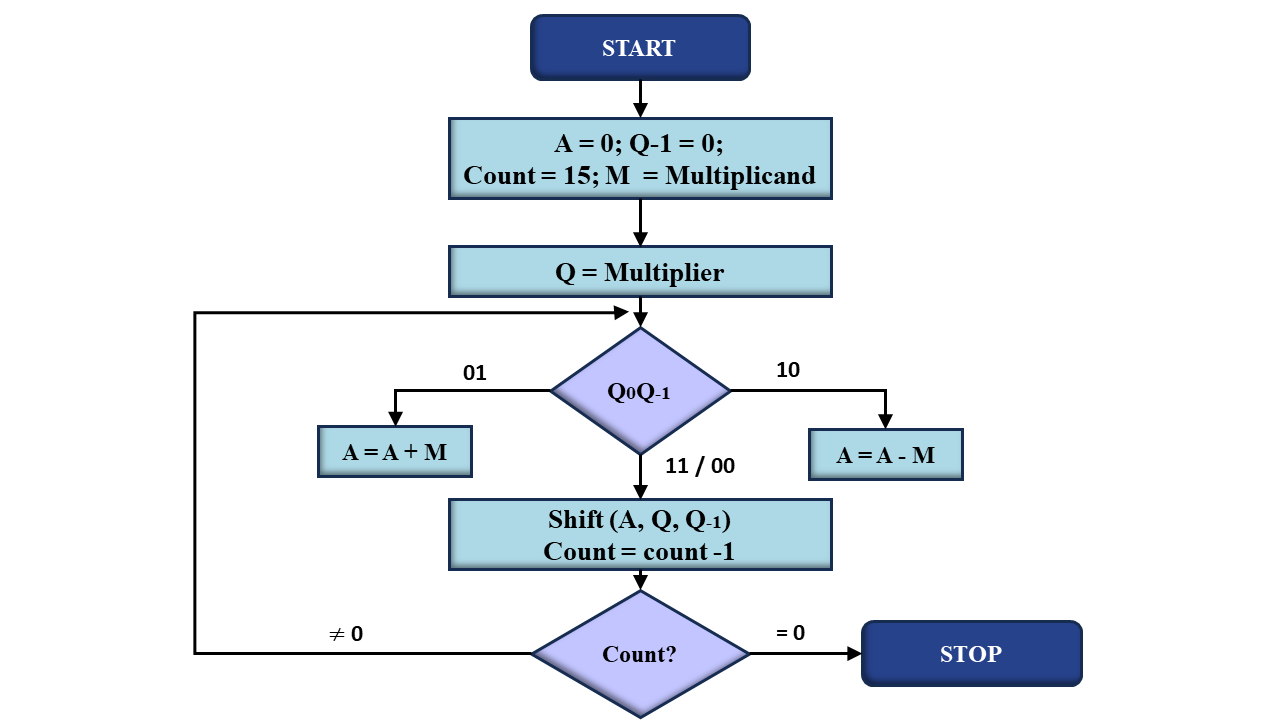


Figure 4. Booths’ Multiplier Algorithm

## **Datapath**

Based on the control signals, the data path performs operation internally. For example, when loadM signal is high, data\_in is transferred to the register-M. Other input signals behave in the similar fashion.

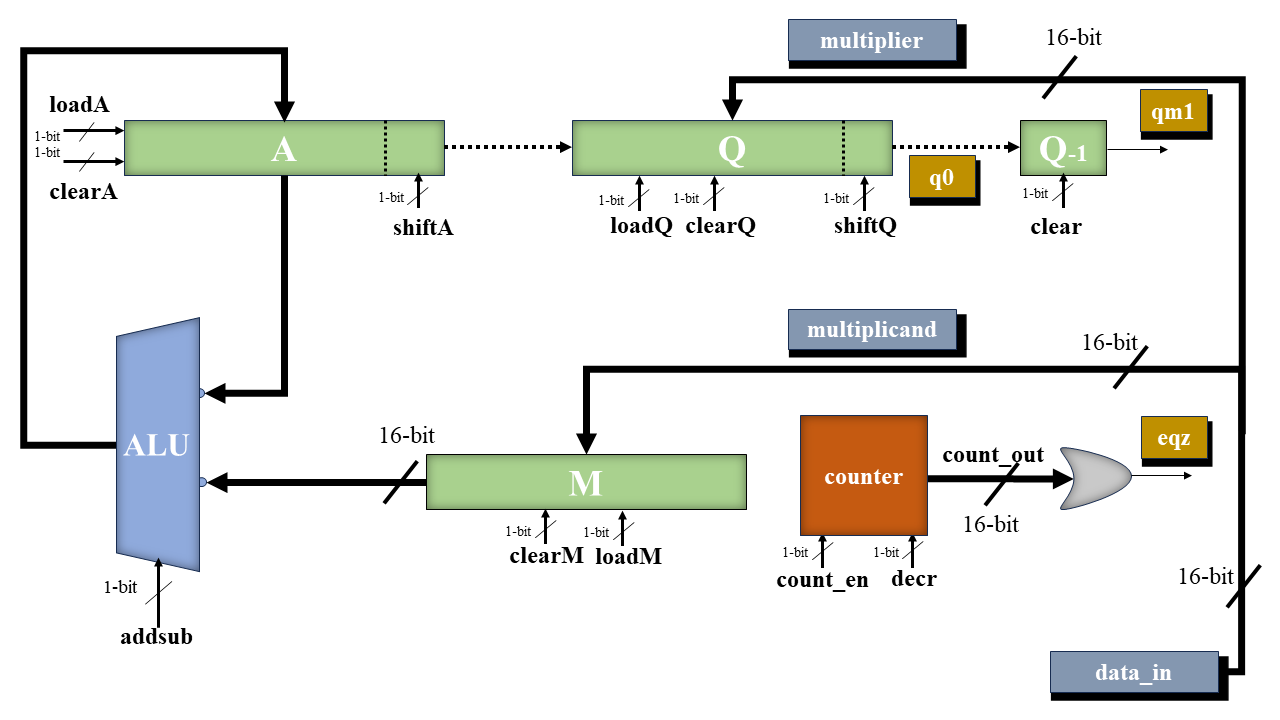


Figure 5. DataPath

**Signal Description**

### **Input Signals**

1. **clk:** It is pulsating synchronizing signal having 50 percent duty cycle. The system is positive clock edge sensitive.
2. **data\_in:** 16-bit input data
3. **loadA:** When HIGH, loads the A register in datapath with ALU output  
   .
4. **clearA:** When HIGH, clears the register-A in datapath.
5. **shiftA:** When HIGH, shifts the content of A register in datapath by 1-bit.
6. **loadQ:** When HIGH, loads the Q register in datapath with register-A output.
7. **clearQ:** When HIGH, clears the register-Q in datapath.
8. **shiftQ:** When HIGH, shifts the content of register-Q in datapath by 1-bit.
9. **loadM:** When HIGH, loads the register-A in datapath with data\_in.
10. **clearM:** When HIGH, clears the register-A in datapath with data\_in.
11. **clearff:** When HIGH, clears the Qm1 register
12. **addSub:** When HIGH, performs addition of contents of register-A and register-M. When LOW, performs subtraction of contents of register-A and register-M.
13. **decr:** When HIGH, signals the counter to decrement
14. **count\_en:** When HIGH, signals the counter to start counting.

### **Output Signals**

1. **eqz:** When HIGH, resembles counting of counter is completed
2. **Qm1:** Content of Qm1 register
3. **Q0:** LSB of Q register