الجامعة الهاشمية
The Hashemite University

The Hashemite University

Faculty of Engineering

Embedded Systems

Phase 1: Single Purpose Processor Using FPGA Signed Multiplier

Prepared by:

Name	ID	Seat
Moath Yousef AlTahrawi	2141099	42
Muhammad Mustafa AlMashayekh	2138237	10

Dr. Dheya Mustafa

Contents

1. ABSTRACT

In this project, we are building a simple embedded system (single-purpose processor) from scratch. This processor performs parameterized signed multiplication between any two numbers using minimal adders, shifters, subtractors, multiplexers, and shifters, without any multipliers.

In phase 1 of the project, we are going to show how the algorithm works, giving you the steps of the execution process to get to the final answer, and then we will translate the algorithm into a combinational logic design, using combinational circuits and registers and other logical components that we need to create the design we want.

This assignment contains the algorithm steps, with its explanation on how it works, and multiple examples with multiple signed (positive or negative) on 8-bit inputs, traced step-by-step, (ii) 2 negative 8bit number and (iii) two different sign 8bit number to insure that this algorithm works, also you will find the design of the Finite state machine (both before and after optimization) , FSMD, Control unit with each of its status signals, and the Datapath in combinational logic structure.

The first assignment does not contain any code, just the design and the manual implementation, where every picture in this documentation was drawn by one of the team members to ensure their correctness of them.

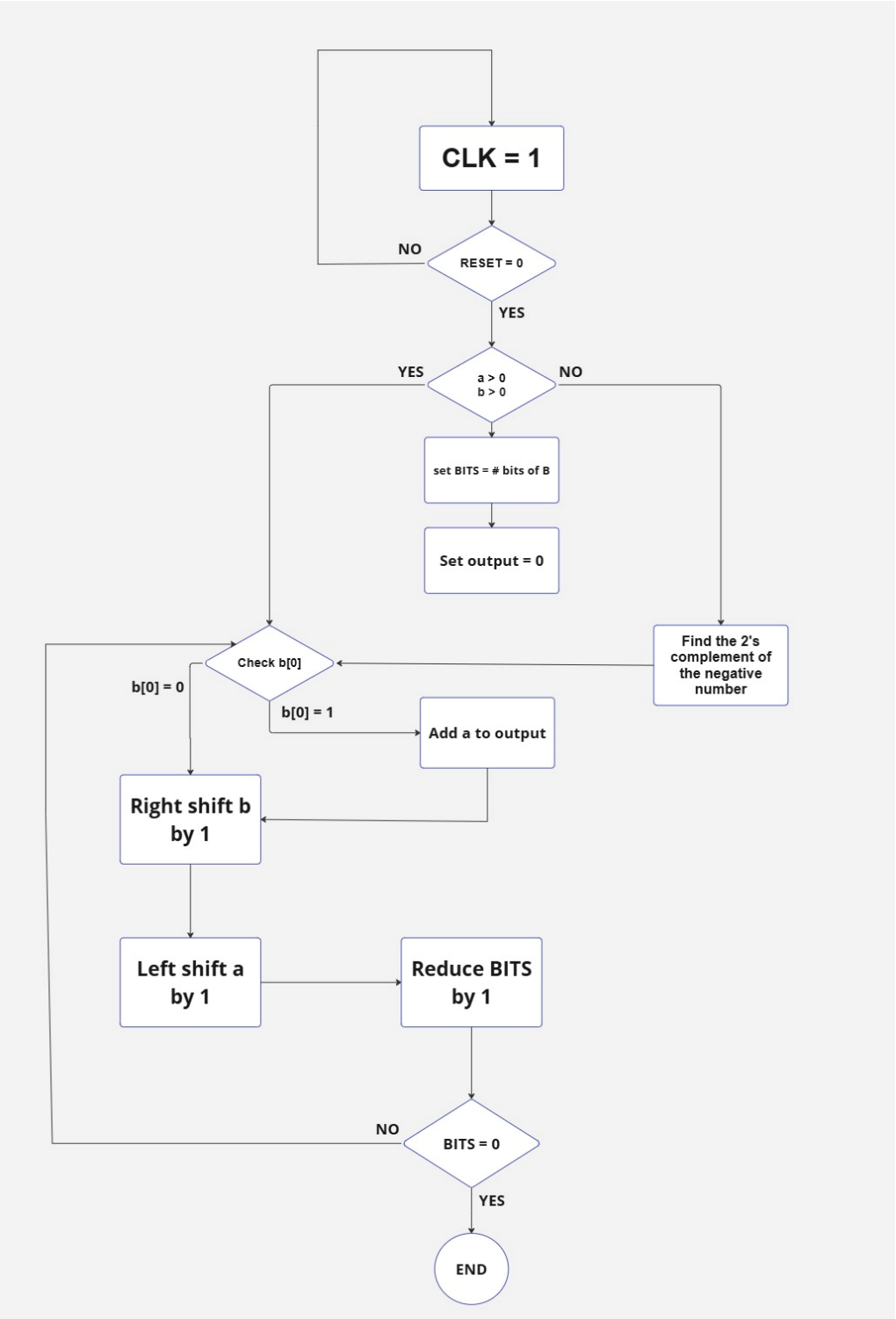
2. The Algorithm

2.1. How the Algorithm Works:

This algorithm uses adders and shifters and comparators to implement the signed multiplication process, the figure below shows the steps of the algorithm as explained below:

a) Start the system, load A and B, and reset the CLK to the first cycle.
b) Check each number to locate whether they are positive or negative and initialize the register (BITS) to the number of bits of the numbers, and the register (OUTPUT) to zero.
c) If there is a negative number, apply the 2's complement to it.
d) Check the first bit in (B), if it is equal to 0, shift right b by 1, and shift left a by 1, if it is equal to 1, add A to the OUTPUT, then do the shifting process mentioned before.
e) Decrement the number of BITS, then check if it is equal to zero, if yes end the process and the OUTPUT is complete, if no, then repeat point **d**.
f) IF A and B have opposite signs THEN OUTPUT = 2's complement of OUTPUT.

2.2. Flowchart of the Algorithm



2.3. Algorithm's Psuedocode:

```
input a, b;
output c = 0;
BITS = # bits of b
if a<0:
        find 2's complement of a;
if b<0:
        find 2's complement of b;

for (i=0,i++,i<=BITS):
        if b[0] == 1:
                c = c + a;
                a = a<<1;
                b = b>>1;
                BITS = BITS - 1
        else:
                a = a<<1;
                b = b>>1;
                BITS = BITS - 1

                BITS = BITS - 1
if a's sign != b's sign:
        c = 2's complement of c;
```

3. Mathematical Examples:

3.1. Multiplying Positive with Positive:

- $a = 15 = \ 0000\ 1111_2$
  $b = 12 = \ 0000\ 1100_2$
  $c = 0 = \ 0000\ 0000\ 0000\ 0000_2$
  $BITS = 8$

Because b [0] = 0 we will shift a once to the left and b once to the right and decrement the value of BITS by 1 which will give us the following result:

- $a = 30 = \ 0001\ 1110_2$
  $b = 6 = \ 0000\ 0110_2$
  $c = 0 = \ 0000\ 0000\ 0000\ 0000_2$
  $BITS = 7$

Because b [0] = 0 we will shift a once to the left and b once to the right and decrement the value of BITS by 1 which will give us the following result:

- $a = 60 = \ 0011\ 1100_2$
  $b = 3 = \ 0000\ 0011_2$
  $c = 0 = \ 0000\ 0000\ 0000\ 0000_2$
  $BITS = 6$

Because b [0] = 1, we will add the value of "a" to c and then we will shift (a) once to the left and (b) once to the right and decrement the value of BITS by 1 which will give us the following result:

- $a = 120 = \ 0111\ 1000_2$
  $b = 1 = \ 0000\ 0001_2$
  $c = 60 = \ 0000\ 0000\ 0011\ 1100_2$
  $BITS = 5$

Because b [0] = 1, we will add the value of "a" to c and then we will shift (a) once to the left and (b) once to the right and decrement the value of BITS by 1 which will give us the following result:

- $a = 240 = \ 1111\ 1000_2$
  $b = 1 = \ 0000\ 0000_2$
  $c = 180 = \ 0000\ 0000\ 1011\ 0100_2$
  $BITS = 4$

Since the value of (b) became zero, we will stop the algorithm here because the value of (c) won't change again

3.2. Multiplying Negative with Negative:

- $a = -15 \ = \ 1111\ 0001_2$
  $b = -12 = \ 1111\ 0100_2$
  $c = 0 = \ 0000\ 0000\ 0000\ 0000_2$
  $BITS = 8$

Because the values (a) and (b) are negative, first we must find the 2's complement of the negative values:

- $a = 15 \ = \ 0000\ 1111_2$
  $b = 12 = \ 1111\ 1100_2$

Which will change the inputs to:

- $a = 15 \ = \ 0000\ 1111_2$
  $b = 12 = \ 1111\ 1100_2$
  $c = 0 = \ 0000\ 0000\ 0000\ 0000_2$
  $BITS = 8$

Because b [0] = 0 we will shift a once to the left and b once to the right and decrement the value of BITS by 1 which will give us the following result:

- $a = 30 = \ 0001\ 1110_2$
  $b = 6 = \ 0000\ 0110_2$
  $c = 0 = \ 0000\ 0000\ 0000\ 0000_2$
  $BITS = 7$

Because b [0] = 0 we will shift a once to the left and b once to the right and decrement the value of BITS by 1 which will give us the following result:

- $a = 60 = \ 0011\ 1100_2$
  $b = 3 = \ 0000\ 0011_2$
  $c = 0 = \ 0000\ 0000\ 0000\ 0000_2$
  $BITS = 6$

Because b [0] = 1, we will add the value of "a" to c and then we will shift (a) once to the left and (b) once to the right and decrement the value of BITS by 1 which will give us the following result:

- $a = 120 = \ 0111\ 1000_2$
  $b = 1 = \ 0000\ 0001_2$
  $c = 60 = \ 0000\ 0000\ 0011\ 1100_2$
  $BITS = 5$

Because b [0] = 1, we will add the value of "a" to c and then we will shift (a) once to the left and (b) once to the right and decrement the value of BITS by 1 which will give us the following result:

- $a = 240 = \ 1111\ 1000_2$
  $b = 1 = \ 0000\ 0000_2$
  $c = 180 = \ 0000\ 0000\ 1011\ 0100_2$
  $BITS = 4$

Since the value of (b) became zero, we will stop the algorithm here because the value of (c) won't change again.

3.3. Multiplying Positive with Negative:

- $a = 15 \ = \ 0000\ 1111\ _2$
  $b = -12 = \ 1111\ 0100_2$
  $c = 0 = \ 0000\ 0000\ 0000\ 0000_2$
  $BITS = 8$

Because the values (b) are negative, first we must find the 2's complement it first:

$$b = 12 = \ 1111\ 1100_2$$

Which will change the inputs to:

- $a = 15 = \ 0000\ 1111_2$
  $b = 12 = \ 0000\ 1100_2$
  $c = 0 = \ 0000\ 0000\ 0000\ 0000_2$
  $BITS = 8$

Because b [0] = 0 we will shift a once to the left and b once to the right and decrement the value of BITS by 1 which will give us the following result:

- $a = 30 = \ 0001\ 1110_2$
  $b = 6 = \ 0000\ 0110_2$
  $c = 0 = \ 0000\ 0000\ 0000\ 0000_2$
  $BITS = 7$

Because b [0] = 0 we will shift a once to the left and b once to the right and decrement the value of BITS by 1 which will give us the following result:

- $a = 60 = \ 0011\ 1100_2$
  $b = 3 = \ 0000\ 0011_2$
  $c = 0 = \ 0000\ 0000\ 0000\ 0000_2$
  $BITS = 6$

Because b [0] = 1, we will add the value of "a" to c and then we will shift (a) once to the left and (b) once to the right and decrement the value of BITS by 1 which will give us the following result:

- $a = 120 = \ 0111\ 1000_2$
  $b = 1 = \ 0000\ 0001_2$
  $c = 60 = \ 0000\ 0000\ 0011\ 1100_2$
  $BITS = 5$

Because b [0] = 1, we will add the value of "a" to c and then we will shift (a) once to the left and (b) once to the right and decrement the value of BITS by 1 which will give us the following result:

- $a = 240 = \ 1111\ 1000_2$
  $b = 1 = \ 0000\ 0000_2$
  $c = 180 = \ 0000\ 0000\ 1011\ 0100_2$
  $BITS = 4$

Since the value of (b) became zero, we will stop the algorithm here because the value of (c) won't change again. Because the values of (a) and (b) have different signs we need to find the 2's complement of the output c:

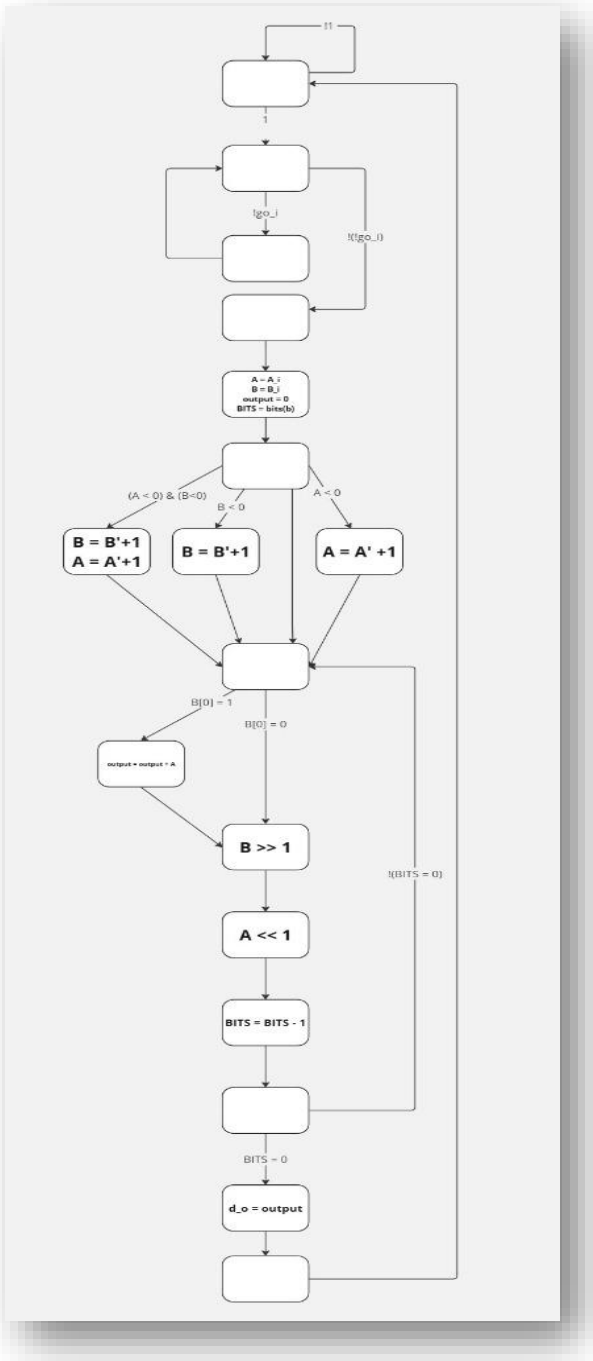
$$c = 180 = \ 0000\ 0000\ 1011\ 0100_2$$

Will become:

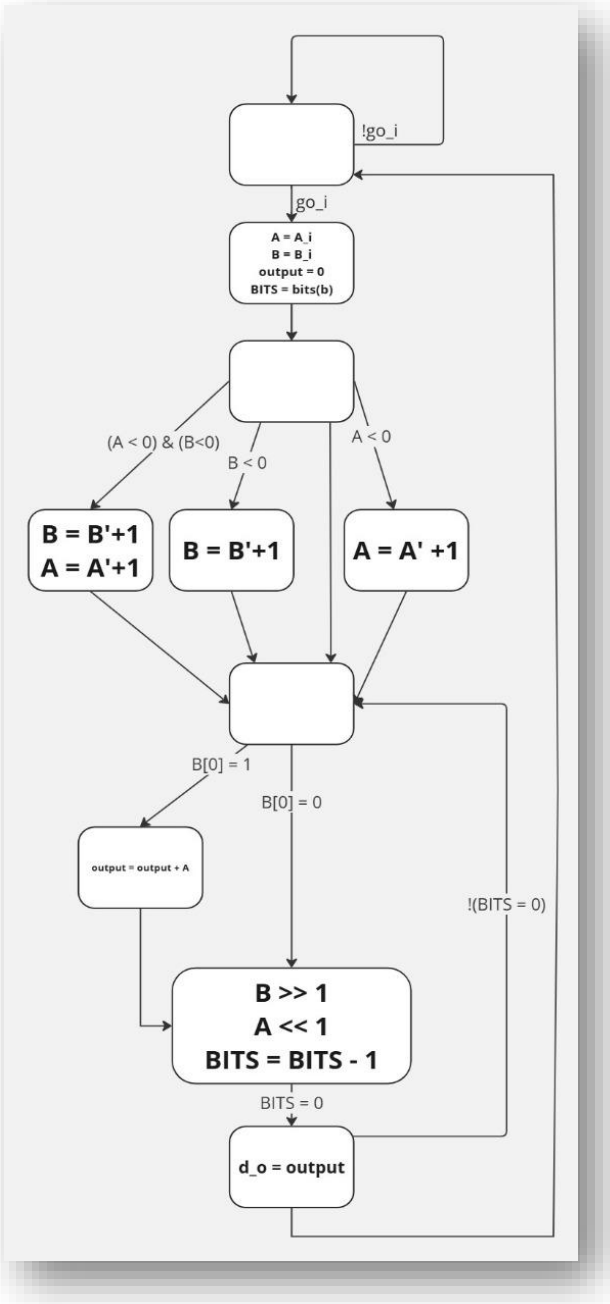
$$c = -180 = \ 1111\ 1111\ 0100\ 1100_2$$

4. Finite State Machine Designs:

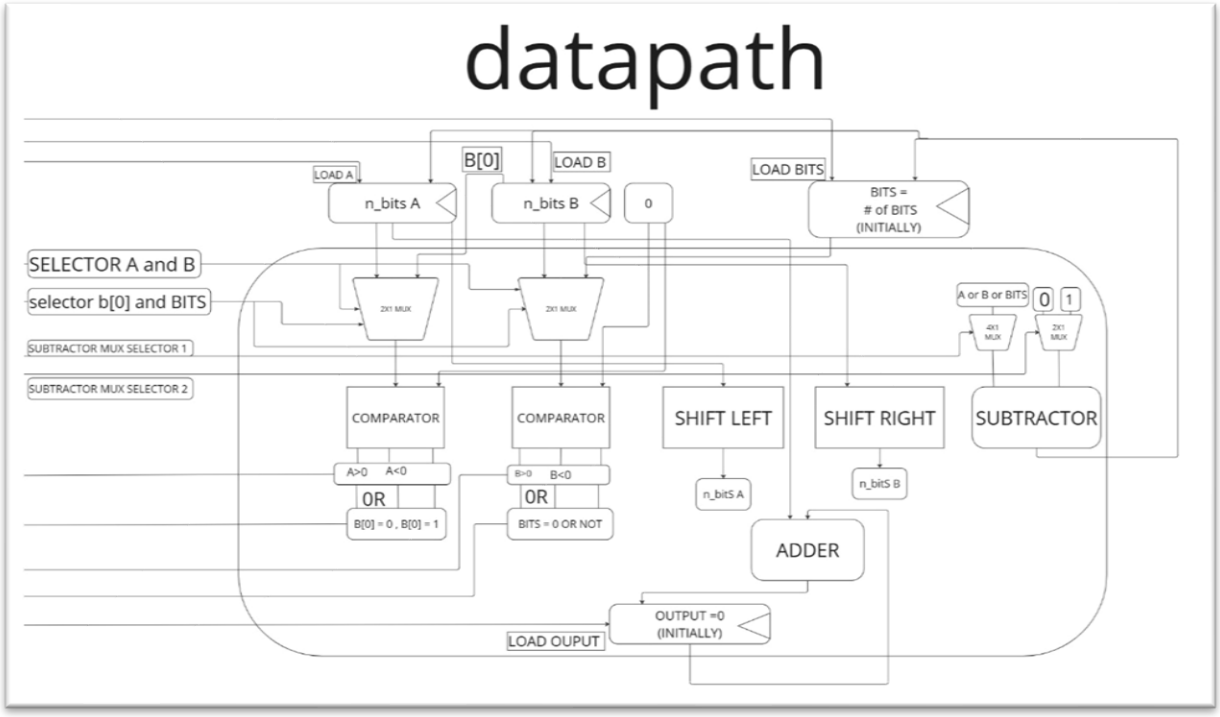
a) Before Optimization:



b) After Optimization:



5. Datapath Design:



This data path consists of multiple components:

a) **Registers (n_bits A, B, BITS, and OUTPUT)**:

- **n_bits A, n_bits B**: Holds the numbers.
- **BITS**: Tracks the number of bits left to process.
- **OUTPUT**: Is initialized to zero, tracks the value of the output through the process and accumulates the result.

b) **MUX Selectors and Load signals**:

- Control the selection between inputs and allow conditional logic to direct operations and allow registers to update their values. (from the control unit)

c) **Comparator Blocks**:

- Compare values to determine conditions and give results as status signals to the control units.

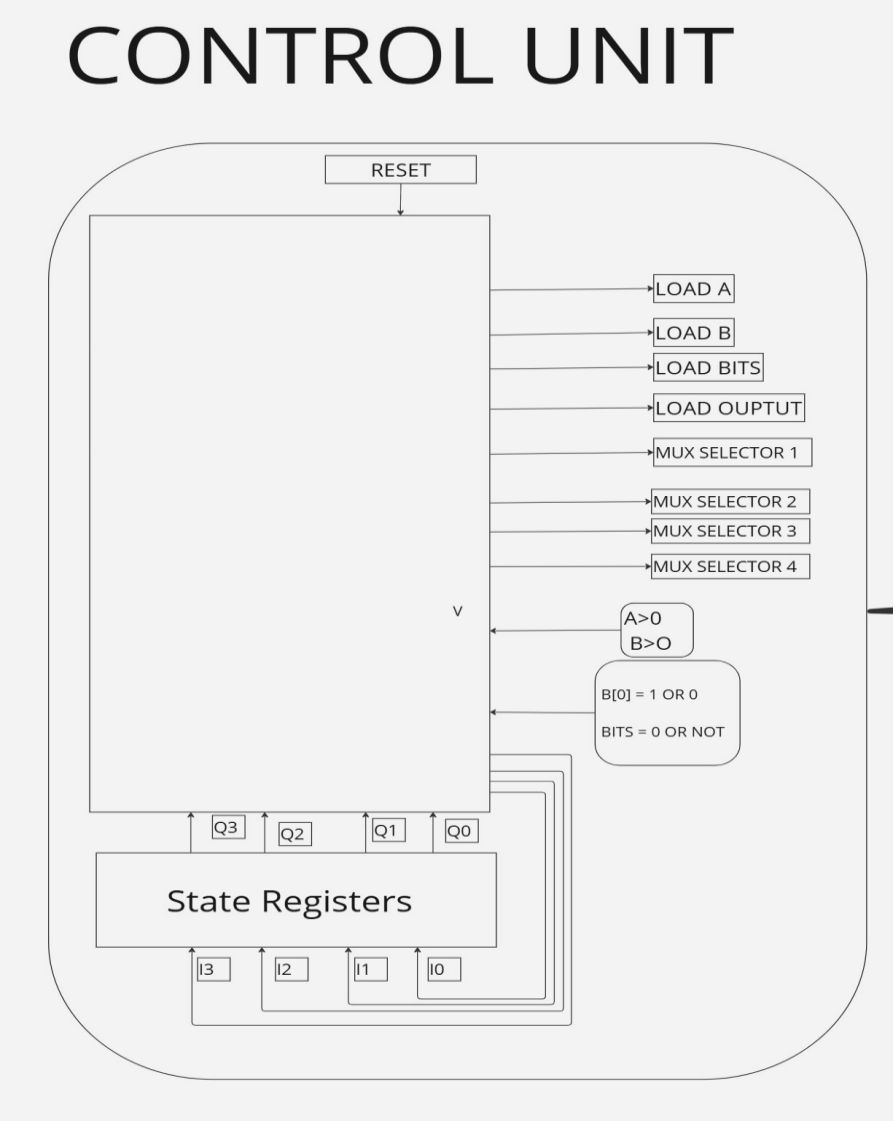
d) **Shift Operations**:

- **SHIFT LEFT** and **SHIFT RIGHT** modules perform arithmetic shifts needed for the Algorithm.

e) **Arithmetic Units**:

- **SUBTRACTOR**: Computes both the decrement and the complement (if needed) as part of the encoding.
- **ADDER**: Combines results during iterations for the partial products.

6. Control Unit:



This control unit consists of multiple inputs and outputs:

a) Load signals: allow the registers to save the values given to them
b) State registers: store the current state of the control unit
c) Input Conditions and Comparisons
d) RESET Input to initialize the control unit

7. Whole Design: