Assignment No.: 05

Title: Implement Booth Multiplication algorithm.

Aim: Build a small compute cluster using Raspberry Pi/BBB modules to implement booth Multiplication algorithm.

Objectives:

To understand working of Cluster of BBB (BeagleBone Black)

Implement booths multiplication algorithm.

Theory:

Booth's algorithm can be implemented by repeatedly adding (with ordinary unsigned binary addition) one of two predetermined values A and S to a product P, then performing a rightward arithmetic shift on P.

Let m and r be the multiplicand and multiplier, respectively; and let x and y represent the number of bits in m and r.

1. Determine the values of A and S, and the initial value of P. All of these numbers should have a length equal to (x + y + 1).

    1. A: Fill the most significant (leftmost) bits with the value of m. Fill the remaining (y + 1)bits with zeros.

    2. S: Fill the most significant bits with the value of (−m) in two's complement notation. Fill the remaining (y + 1) bits with zeros.

    3. P: Fill the most significant x bits with zeros. To the right of this, append the value of r. Fill the least significant (rightmost) bit with a zero.

2. Determine the two least significant (rightmost) bits of P.

    1. If they are 01, find the value of P + A. Ignore any overflow.

    2. If they are 10, find the value of P + S. Ignore any overflow.

    3. If they are 00, do nothing. Use P directly in the next step.

    4. If they are 11, do nothing. Use P directly in the next step.

3. Arithmetically shift the value obtained in the 2nd step by a single place to the right. Let P now equal this new value.

4. Repeat steps 2 and 3 until they have been done y times. Drop the least significant (rightmost) bit from P. This is the product of m and r.

We will explain Booth's Multiplication with the help of following **example**

Multiply 14 times -5 using 5-bit numbers (10-bit result).

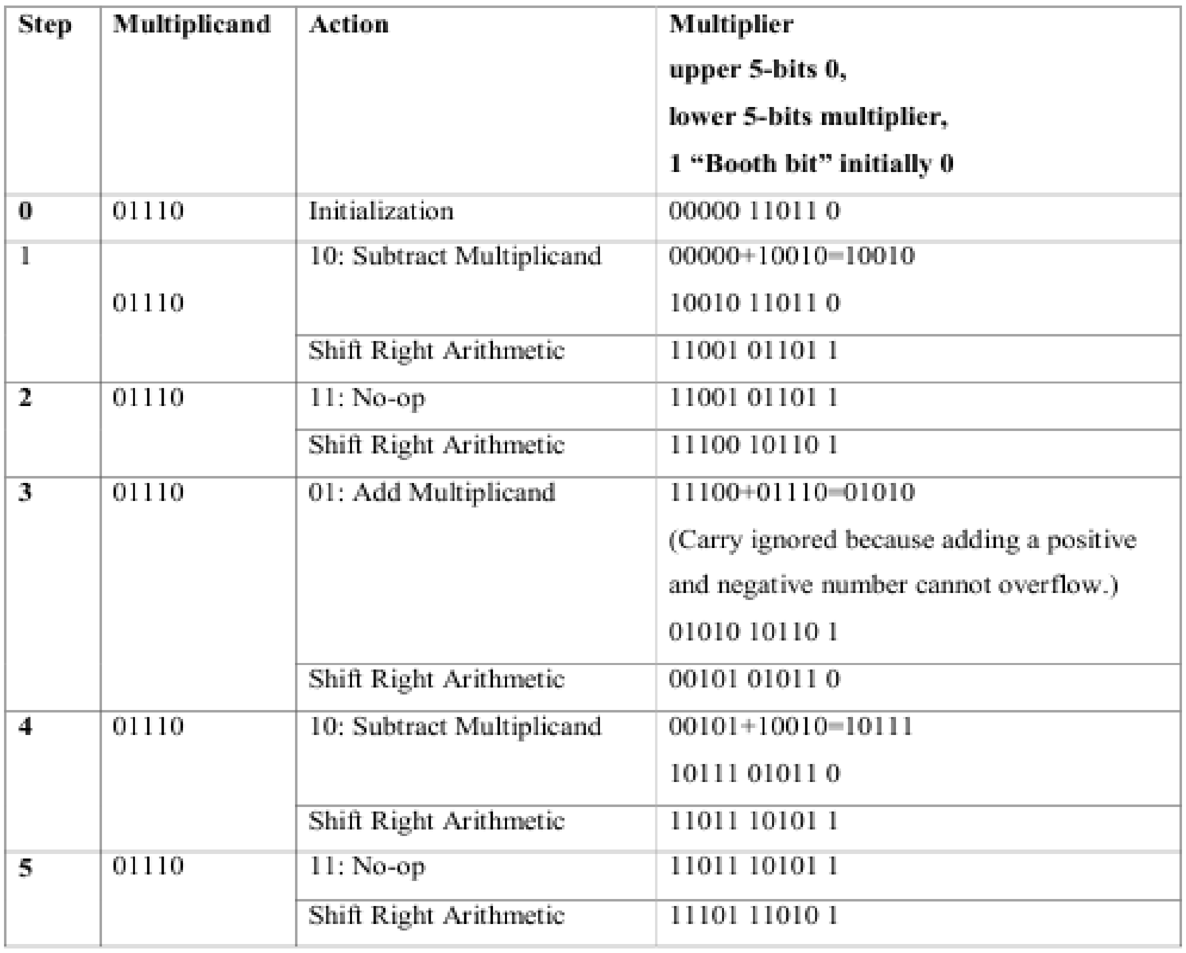
14 in binary: 01110

-14 in binary: 10010 (so we can add when we need to subtract the multiplicand)

-5 in binary: 11011

Expected result: -70 in binary: 11101 11010

The Multiplication Process is explained below:



Step	Multiplicand	Action	Multiplier upper 5-bits 0, lower 5-bits multiplier, 1 "Booth bit" initially 0
0	01110	Initialization	00000 11011 0
1	01110	10: Subtract Multiplicand	00000+10010=10010 10010 11011 0
		Shift Right Arithmetic	11001 01101 1
2	01110	11: No-op	11001 01101 1
		Shift Right Arithmetic	11100 10110 1
3	01110	01: Add Multiplicand	11100+01110=01010 (Carry ignored because adding a positive and negative number cannot overflow.) 01010 10110 1
		Shift Right Arithmetic	00101 01011 0
4	01110	10: Subtract Multiplicand	00101+10010=10111 10111 01011 0
		Shift Right Arithmetic	11011 10101 1
5	01110	11: No-op	11011 10101 1
		Shift Right Arithmetic	11101 11010 1

Booth's Multiplication algorithm is an elegant approach to multiplying signed numbers. Using the standard multiplication algorithm, a run of 1s in the multiplier in means that we have to add as many successively shifted multiplicand values as the number of 1s in the run.

```
     0010two

x    0111two

 --------
```

```
+   0010 multiplicand shifted by 0 bits left

+   0010 multiplicand shifted by 1 bit left

+   0010 multiplicand shifted by 2 bits left

+ 0000

  --------

  00001110two
```

We can rewrite 2i - 2i-j as:

2i - 2i-j = 2i-j x (2j - 1)

= 2i-j x (2j-1 + 2j-2 + ... + 20)

= 2i-1 + 2i-2 + ... + 2i-j

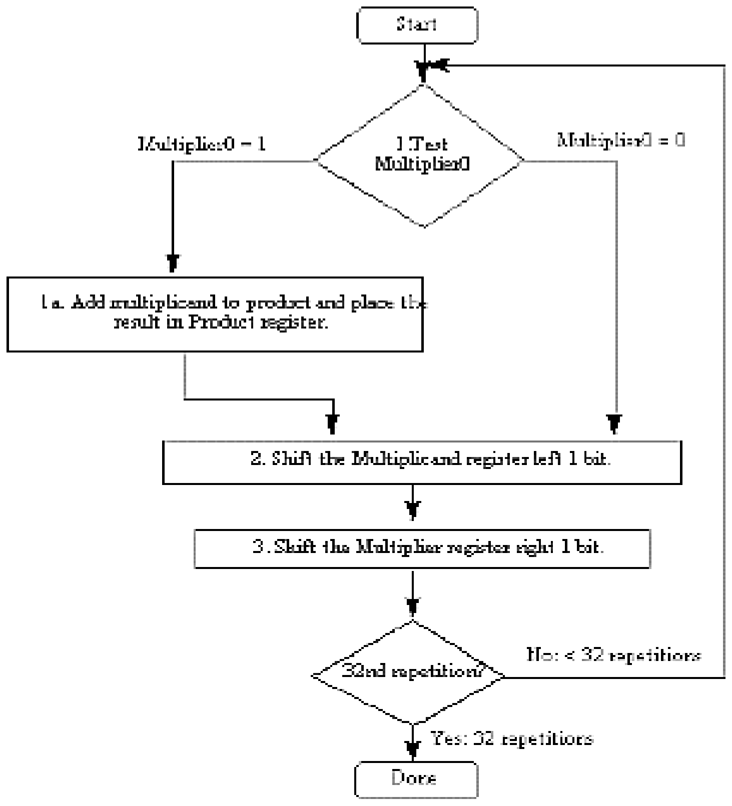
For example 0111two = 23ten - 20ten. So 0010two x 0111two can be written as:

0010two x (1000two - 0001two)

Or to make it look like the multiplication before:

```
     0010two

x    0111two

  --------

-    0010 = 0010two x - 0001two

+    0000

+    0000

+    0010     = 0010two x + 1000two
00001110two
```

Flowchart of booth’s multiplication Algorithm :


Start
1.Test Multiplier0
Multiplier0 = 1
Multiplier0 = 0
1a. Add multiplicand to product and place the result in Product register.
2. Shift the Multiplicand register left 1 bit.
3. Shift the Multiplier register right 1 bit.
32nd repetition?
No: < 32 repetitions
Yes: 32 repetitions
Done

Multiply Algorithm

Mathematical Model:

Let, S be the System Such that,

A={S, E, I, O, F, DD, NDD, F_min, F_fri, CPU_Core, Mem_Shared, success, failure} Where,

S= Start state,

E= End State,

I= Set of Input

O= Set of Out put

F =Set of Function

DD=Deterministic Data

NDD=Non Deterministic Data

F_Min=Main Function

F_Fri= Friend Function

CPU_Core= No of CPU Core.

Mem_Shared=Shared Memory.

Function:

1) Splitting Function = this function is used for splitting unsorted list.

2) Sorting Function = this function is used for sorting list.

3) Binary Search = this function apply binary search on sorted list.

Success Case: It is the case when all the inputs are given by system are entered correctly. Failure Case: It is

the case when the input does not match the validation Criteria.

Conclusion: Thus we have implemented booths algorithm using small cluster of BBB.

Steps to be followed for successful exection of program

Prerequisites:

First disable the firewall on the machine used for running server code.

Check the IP addres and port number in socket's connect() and bind() functions in all the files.

Make sure all the code have the IP and port number of the machine running the server code.

Steps:

Run the server code first and enter the multiplicand an multiplier in integer base 10 format.(The usual numbers like -5, 3 ,4...)

Run the multiplicand code next on another machine.

Run the multiplier code on yet another machine.

1. The multiplicand code will display the values of A and S. It will also display the packed string which is sent back to the server.

2. The multiplier code will display the value of P

3. The server code will display the connection received from both the clients. Finally, it also outputs the answer in binary format.

Code:

```
server_booth.py
# take values A,S from clients and run the main loop for calculating P

import socket

def addition(op1, op2):
        # length of P and A and S is same.
        result=""
        carry="0"
        for i in range(len(op1)-1, -1, -1):         #run reverse loop
                if op1[i]=="1" and op2[i]=="1":
                        if carry=="1":
                                result="1"+result
                                carry="1"
                        else:                                   #carry = 0
                                result="0"+result
                                carry="1"

                elif op1[i]=="0" and op2[i]=="0":
                        if carry=="1":
                                result="1"+result
                                carry="0"
                        else:                                   #carry = 0
                                result="0"+result
                                carry="0"

                elif op1[i]=="0" and op2[i]=="1":
                        if carry=="1":
                                result="0"+result
                                carry="1"
                        else:                                   #carry = 0
                                result="1"+result
                                carry="0"

                else:                                           # 1, 0
                        if carry=="1":
                                result="0"+result
                                carry="1"
                        else:                                   #carry = 0
                                result="1"+result
                                carry="0"
        return result

s = socket.socket()         # Create a socket object
s.bind(("192.168.6.80", 9001))        # Bind to the port


M=int(input("Enter a multiplicant:"))
R=int(input("Enter a  multiplier:"))
M, R=bin(M), bin(R)
print "Binary representation: ", M, R

s.listen(2)                  # Now wait for client connection.
client, addr = s.accept()    # Establish connection with client.
print 'Got connection from', addr

client2, addr2 = s.accept()
print 'Got connection from', addr2
```

```
'''
Send the value of both. Client will return A, S. It will also return length_R as first param.
<Length_R>A<A>S<S>
Send the value of length of R and value of R. Client will return P.      P<P>
'''
client.send(M+" "+R)

AS=client.recv(1024)  # recv A, S
index_A=AS.index("A")
index_S=AS.index("S")
A=AS[index_A+1:index_S]
S=AS[index_S+1:]

length_R=int(AS[:index_A])
client2.send(str(length_R)+" "+R)
P=client2.recv(1024)  # recv P
index_P=P.index("P")
P=P[index_P+1:]
P_length=len(P)

#we've got A,S,P in strings
for i in range(length_R):

        last_two_digits=P[P_length-2:P_length]

        if last_two_digits == "01":
                #add A in P and store that in P and ignore overflows
                P=addition(A, P)
        elif last_two_digits == "10":
                #add S in P aND store the result in P and IGNORE OVerflows
                P=addition(S, P)

        #print "After addn", P
        #arithmetic right shift (copy the sign bit as well). Start looping from the right
most digits
        P=P[0]+P[0:P_length-1]

P=P[:P_length-1]
print P


client _multiplier.py
# calculate value P and send it back to server.

import socket

def twos_comp(binM):
        S = [int(x) for x in binM]
        flag = 0
        for i in range(len(S)-1, -1, -1):
                if flag==1:
                        #invert
                        if S[i]==1:
                                S[i]=0
                        else:
                                S[i]=1
                        continue

                if S[i]==1:
                        flag=1
        return S
```

```
s = socket.socket()         # Create a socket object
s.connect(("192.168.6.80", 9001))
temp=s.recv(1024)
temp=temp.split()
length_R, R= int(temp[0]), temp[1]

if R[0]=="-":
	R=R[3:]
	#origR=R
	R=twos_comp(R)
	R=[str(x) for x in R]
	R=''.join(R)
	for i in range (length_R-len(R)):
		R="1"+R
else:
	R=R[2:]
	for i in range (length_R-len(R)):
		R="0"+R
	#flag_R=2

P = []
for i in range(2*length_R + 1):
	P.append(0)

print "check length of P: ", P

for i in range(len(R)):
	P[length_R+i]=int(R[i])

P=[str (x) for x in P]
P="".join(P)
print P
s.send("P"+P)



Client_multiplicand.py
# calculate values A, S and send it back to server.

import socket

def twos_comp(binM):
	S = [int(x) for x in binM]
	flag = 0
	for i in range(len(S)-1, -1, -1):
		if flag==1:
			#invert
			if S[i]==1:
				S[i]=0
			else:
				S[i]=1
			continue

		if S[i]==1:
			flag=1
	return S

s = socket.socket()         # Create a socket object
s.connect(("192.168.6.80", 9001))
temp=s.recv(1024)
temp=temp.split()
M, R=temp[0], temp[1]
```

```
origM, origR="", ""
Max_length=0

flag,flag_R=0,0                     # flag=1: -M, flag=2: M. flag_R=1: -R, flag_R=2: R
if M[0]=="-":
       M=M[3:]
       origM=M
       M=twos_comp(M)
       M=[str(x) for x in M]
       M=''.join(M)
       flag=1
else:
       M=M[2:]
       flag=2

if R[0]=="-":
       R=R[3:]
       origR=R
       R=twos_comp(R)
       R=[str(x) for x in R]
       R=''.join(R)
       flag_R=1
else:
       R=R[2:]
       flag_R=2

if len(M)>= len(R):
       padding=len(M)-len(R)+1 #+1 for sign bit
       if flag==1:
              M="1"+M
       else:
              M="0"+M
       for i in range (padding):
              if flag_R==1:
                     R="1"+R
              else:
                     R="0"+R
       Max_length=len(M)
else:
       padding=len(R)-len(M)+1
       if flag_R==1:
              R="1"+R
       else:
              R="0"+R
       for i in range (padding):
              if flag==1:
                     M="1"+M
              else:
                     M="0"+M
       Max_length=len(R)

print M, R

#now calc A, S using the length of M and R and 1 (lenM+lenR+1)
A = []
for i in range(len(M)+len(R)+1):
       A.append(0)

for i in range(len(M)):
       A[i]=int(M[i])
A=[str(x) for x in A]
print "A: ", A
#A is ready at this point
```

```
if flag==1:           # orignal M was -ve. So we need origM with the minus sign eliinated
       for i in range(Max_length-len(origM)):
              origM="0"+origM
       S=[str(x) for x in origM]

else:
       S=twos_comp(M)

for i in range(len(M)+len(R)+1-len(S)):
       S.append(0)

S=[str(x) for x in S]

#S is ready at this point
print "S: ", S

#pack the A ans S in a buffer string
Send_AS= str(len(R))+"A"+''.join(A) #secret- length of both operands is same. So u can re-
place R with M
Send_AS += "S"+''.join(S)
print Send_AS

#send the A and S to server and the job here is done
s.send(Send_AS)
```

OUTPUT

```
Server side :

root@Student-301:/home/student/Documents/A-5 Booth's Algorithm# python server_booth.py

Enter a multiplicant:5

Enter a  multiplier:2

Binary representation:  0b101 0b10

Got connection from ('192.168.6.67', 36176)

Got connection from ('192.168.6.79', 36224)

00001010

root@Student-301:/home/student/Documents/A-5 Booth's Algorithm#


--------------------------------------------------------------------------------


Client1 :

root@student-302:/home/student/Documents# python client_multiplier.py check length of P:  [0,
0, 0, 0, 0, 0, 0, 0, 0]

000000100

root@student-OptiPlex-3010:/home/student/Documents#


-------------------------------------------------------------------------------


Client2 :

root@student-303:/home/student/Documents# python client_multiplicand.py

0101 0010

A:  ['0', '1', '0', '1', '0', '0', '0', '0', '0']

S:  ['1', '0', '1', '1', '0', '0', '0', '0', '0']

4A010100000S101100000

root@student-303:/home/student/Documents#
```