Report for Final Lab

Group Members: Yihao Wang (7410178057), Jiahui Zhao(8320271522), Pooja Pawar(7854000155)

Part I Montgomery Multiplication Design

a. Brief Description of Design

In this design, we achieved modular multiplication operation by using Montgomery Reduction Algorithm. In this algorithm, we simplify mod N operation (N is a big number) to mod R (R = 129) which can be implemented by shifting) so we use simple multiplication, addition and subtraction to achieve complex division and then simplify logic.

Firstly, we should implement m = (T*N_i)mod R operation by using one multiplier and shifting operation to calculate T mod R (T = x*y), then another multiplier for calculating m = [T(mod R)]*N_i(mod R). Secondly, we transfer m to S = T*R_i(mod N) by using transfer equation t = (T +mN)/R and generate S based the value of t and N. These are achieved by using one multiplier for mN calculation and RCA for calculating T + mN, ignoring the lower 7bit and treating the upper 8bit as quotient because R = 2^7 = 128. Lastly, we generated S by comparing N and t by using subtraction for comparison

b. Python Script Code

```
"""
    The python script for final lab (part1) of ee577a
    By Yihao Wang
"""
import time
start_time = time.time()

## Coversion between decimal number and binary number
def dec_to_bin (num,width):
    the_num = num
    bin_str = ''
    if (num < 0 or num >= 2 ** width):
            print('the parameter must be [0,' + str(2 ** width - 1) + ']!')
    else:
        for num in range(width):
            bin_str = str(the_num % 2) + ' ' + bin_str
            the_num = int(the_num / 2)
        return bin_str
```

```
## judge if two integers are coprime pair
## Returns True or False
def isCoprime(num_a, num_b) :
    t = 0
    while num_b >0 :
        t = num_a % num_b
        num_a = num_b
        num_b = t
    if num_a == 1 :
        return True
    else :
        return False

## Gets values of sk and rk based on Extended Euclidean Algorithm: a * sk + b * tk = 1
## Returns a list: [sk, tk]
def getEucildean(int_a, int_b) :
    r0 = int_a; r1 = int_b; s0 = 1; s1 = 0; t0 = 0; t1 = 1; q = int(r0 /r1)
    while r1 != 0 :
        r_temp = r1; r1 = r0 -q * r1; r0 = r_temp
        s_temp = s1; s1 = s0 - q * s1; s0 = s_temp
        t_temp = t1; t1 = t0 - q * t1; t0 = t_temp
        if r1 != 0 :
            q = int(r0 / r1)
    return [s0, t0]

## Gnerates random inputs valuse and calculates outputs
## Returns all values by using list: [inputs, outputs]
def getRandom(inputs,outputs) :
    import random
    num_r = (2 ** 7)
    output = [None] * (len(inputs))

    # Generates random N and uses it if N is coprime to R
    num_n = random.randint(0, 2 ** 7 - 1)
    while not(isCoprime(128, num_n)) :
        num_n = random.randint(0, 2 ** 7 - 1)
    output[inputs.index('N')] = num_n

    num_x = random.randint(0, (2 ** 7 - 1))
    num_y = random.randint(0, (2 ** 7 - 1))
    while not((num_x * num_y >= 0) and (num_x * num_y <= (num_r * num_n - 1))) :
        num_x = random.randint(0, (2 ** 7 - 1))
        num_y = random.randint(0, (2 ** 7 - 1))
    output[inputs.index('X')] = num_x
    output[inputs.index('Y')] = num_y
    ## Get values of R^(-1) and N^(-1) based on results of getEucildean method
    list_temp = getEucildean(num_r, num_n)
```

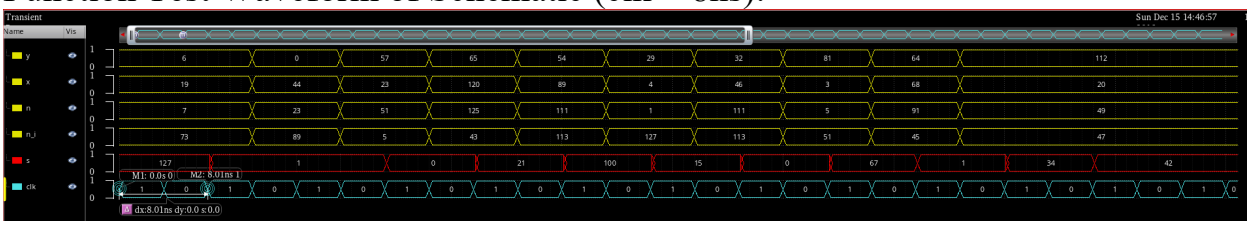
```
    num_r_m1 = 0

    # Generates value of N_inverse based sk and tk given by function getEucildean()
    if (list_temp[0] > 0) and (list_temp[1] < 0) :
        output[inputs.index('N-1')] = -list_temp[1]
        num_r_m1 = list_temp[0]
    else:
        output[inputs.index('N-1')] = -(list_temp[1] - num_r)
        num_r_m1 = list_temp[0] + num_n

    for i in range(len(outputs)) :
        output.append((num_x * num_y * num_r_m1) % num_n)
    return output

## Generates vector file uisng standard vector file format
def vec_generator (inputs = ['X', 'Y', 'N', 'N-1'], inputs_width = [8, 8, 8, 8],
outputs = ['S'], outputs_width = [8],  number = 30, clk = 10, delay = 2, unit = 'ns',
slope = 0.01, vih = 1.8, vil = 0, file_path = '/Users/yihaowang/Desktop', file_name =
'test_for_MM.vec'):
    if (len(inputs) != len(inputs_width)) or (len(outputs_width) != len(outputs)) or
(number < 0) or (clk <= 0) or (delay < 0) or(slope < 0) :
        print('Parameter Error!')
    else :
        import os
        import random

        os.chdir(file_path)
        with open(file_name.split('.')[0] + '_' + str(clk) + str(unit) + '.' +
file_name.split('.')[1], 'w') as the_file, open(file_name.split('.')[0] +
'_golden.txt', 'w') as the_golden :

            print('radix', end = ' ', file = the_file)
            length = 0
            for i in inputs_width :
                length += i
            for i in range(length) :
                print('1', end = ' ', file = the_file)

            print('\nio', end = ' ', file = the_file)
            for i in range(length) :
                print('i', end = ' ', file = the_file)

            print('\nvname', end = ' ', file = the_file)
            for i in range(len(inputs)) :
                for j in range(inputs_width[i]) :
                    print(inputs[i] + '<' + str(inputs_width[i] - j - 1) + '>', end =
' ', file = the_file)
            print('\ntunit ' + unit , file = the_file)
```

```
            print('slope ' + str(slope), file = the_file)
            print('vih ' + str(vih), file = the_file)
            print('vil ' + str(vil), file = the_file)

            time = delay
            for i in range(number) :

                random_num = getRandom(inputs, outputs)

                print('\n' + str(time), end = '    ', file = the_golden)
                print('\n' + str(time), end = '    ', file = the_file)
                for j in range(len(inputs)) :
                    print(inputs[j] + '=' + str(random_num[j]), end = ' ', file =
the_golden)
                    print(dec_to_bin(random_num[j], inputs_width[j]), end = ' ', file
= the_file)
                for j in range(len(outputs)) :
                    print(outputs[j] + '=' + str(random_num[len(inputs) + j]), end =
'', file = the_golden)
                time += clk

            print('Successfully Gnererated!')

vec_generator(number = 10, clk = 8, delay = 4)

end_time = time.time()
print('Run Time: %.6fs'%(end_time - start_time))
```

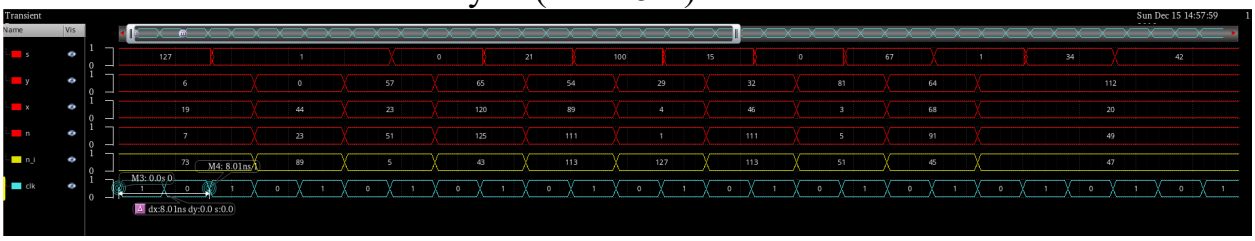
c. Function Test

X(dec)	Y(dec)	N(dec)	N_i(dec)	S(dec)	X(bin)	Y(bin)	N(bin)	N_i(bin)	S(bin)
19	6	7	73	1	0010011	0000110	0000111	1001001	0000001
44	0	23	89	0	0101100	0000000	0010111	1011001	0000000
23	57	51	5	21	0010111	0111001	0110011	0000101	0010101
120	65	125	43	100	1111000	1000001	1111101	0101011	1100100
89	54	111	113	15	1011001	0110110	1101111	1110001	0001111
4	29	1	127	0	0000100	0011101	0000001	1111111	0000000
46	32	111	113	67	0101110	0100000	1101111	1110001	1000011
3	81	5	51	1	0000011	1010001	0000101	0110011	0000000
68	64	91	45	34	1000100	1000000	1011011	0101101	0100010
20	112	49	47	42	010100	1110000	0110001	0101111	0101010

Function Test Waveform of Schematic (clk = 8ns):

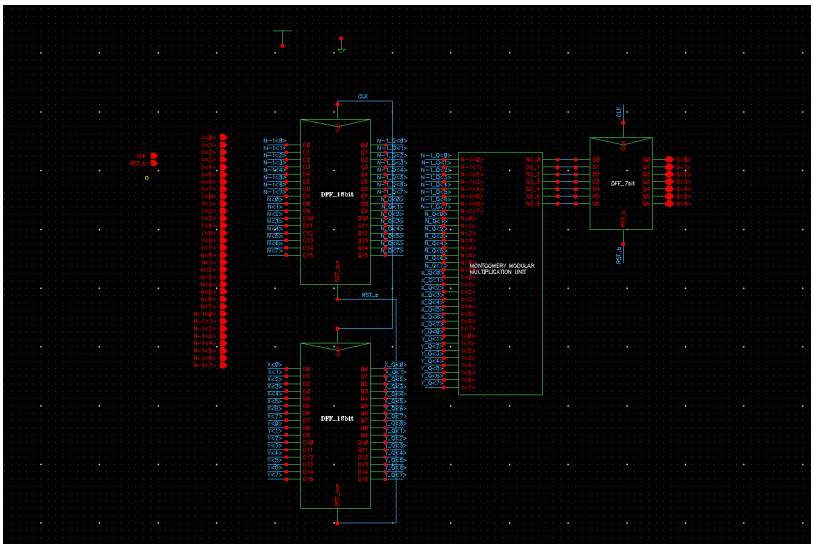


Function Test Waveform of Layout(clk = 8ns):

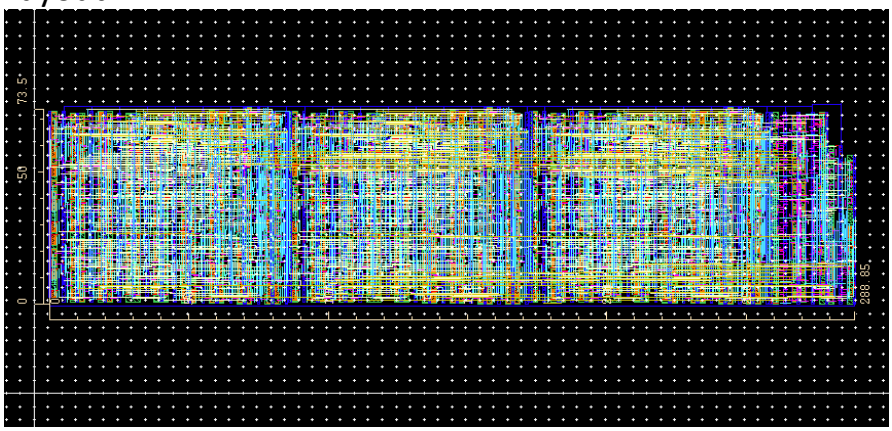


d. Screenshots

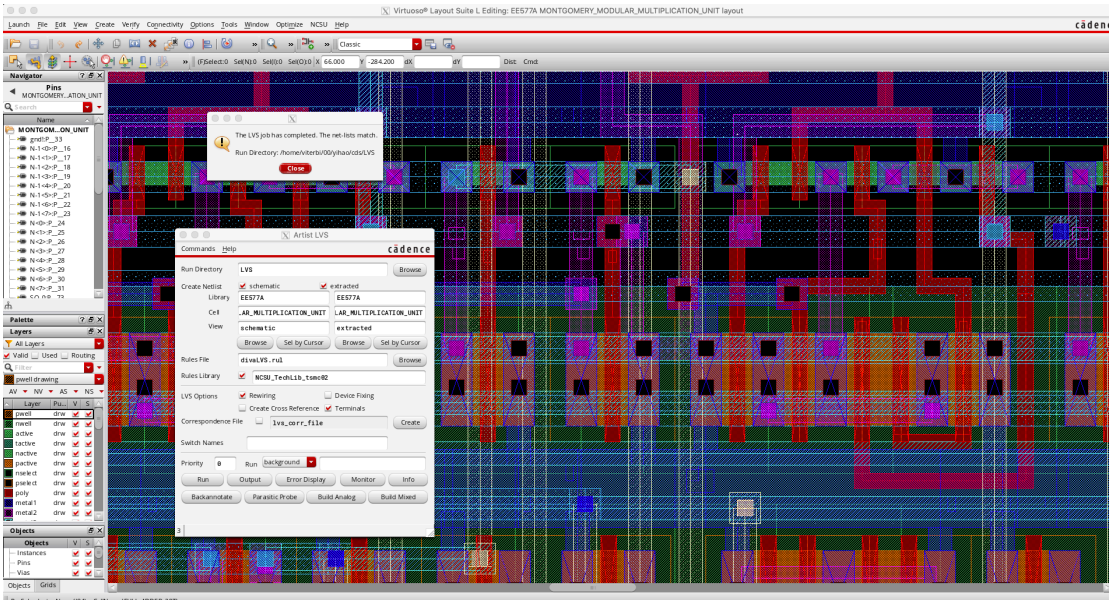
Schematic:



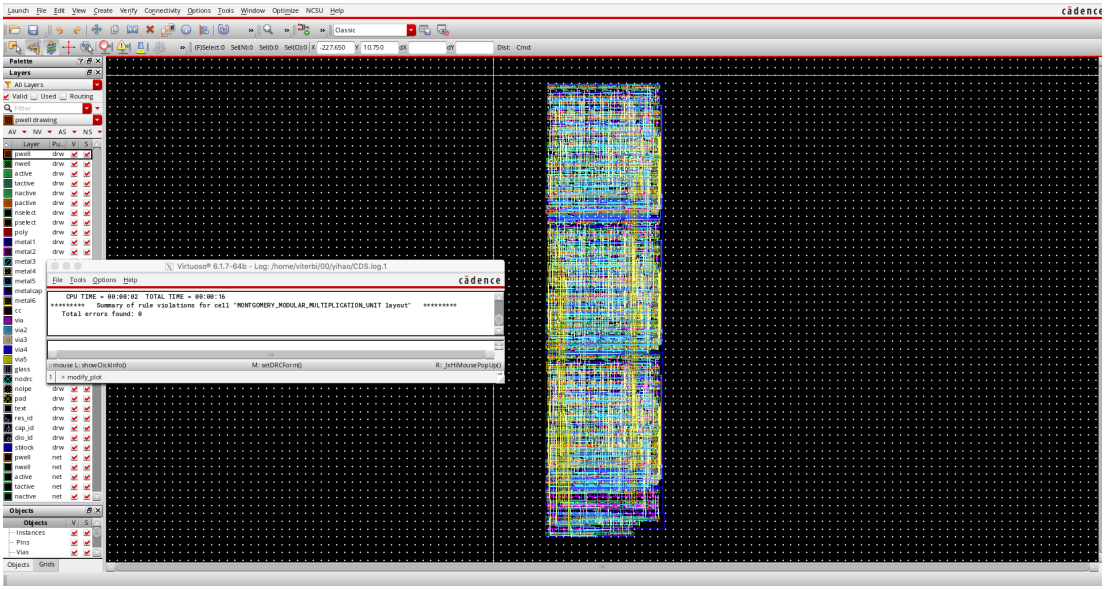
Layout:



LVS Match:



DRC:



e. Key Parameters

Height: 288.85um

Width: 73.5um

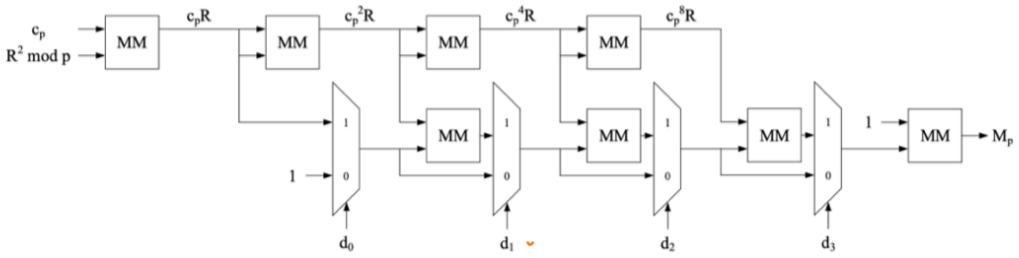
Area: 21230um^2

Minimum Clock: 8ns

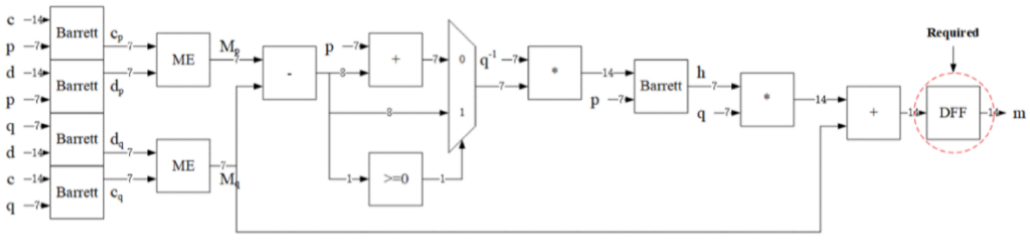
Part II RSA Cryptography System

a. Brief Description of Design

In this design, we implement a decryption unit of cryptography system. It will help restore the plain text m based on cipher text c and private key n, d.The main function is to do calculation of m = Mq + q(q_i(Mp- Mq)mod p). The key part of this calculation is to calculate Mp = (cp^dp)mod p and Mq = (cq^dq)mod q which is called Montgomery Exponential (ME). We implement ME block by using 14 MM (Modular Multiplication Unit) because d is 7bit unsigned number. The upper 7 MM blocks are used to generate (cp*R)mod p, (cp^2 * R)mod p, (cp^4 * R)mod p and (cp^8 * R)mod P and (cp ^ dp)mod p is generated by using MUX and lower 6 MM blocks to generated a combination of former number and using the last MM to eliminate the factor R of output.



Using modulo operation unit, we can easily get dp = d mod(p-1), dq = d mod(q-1), cp = c mod p and cq = c mod q and ME will calculate the value of Mp and Mq. To be noted, when we generate (Mp – Mq), if the difference is negative, we will add a q to this difference to make the difference positive. Lastly , we generate the final outputs m by using related 14bit addition, multiplication and Barret Modulo Unit.



b. Python Script Code

```
"""
    The python script for final lab (part2) of ee577a
    By Yihao Wang
"""
import time
start_time = time.time()

## Coversion between decimal number and binary number
def dec_to_bin (num,width):
    the_num = num
    bin_str = ''
    if (num < 0 or num >= 2 ** width):
            print('the parameter must be [0,' + str(2 ** width - 1) + ']!')
    else:
        for num in range(width):
            bin_str = str(the_num % 2) + ' ' + bin_str
            the_num = int(the_num / 2)
        return bin_str

## Judges if two integers are coprime pair
## Returns True or False
def isCoprime(num_a, num_b) :
    t = 0
    while num_b >0 :
        t = num_a % num_b
        num_a = num_b
        num_b = t
    if num_a == 1 :
        return True
    else :
        return False

## Gets values of sk and rk based on Extended Euclidean Algorithm: a * sk + b * tk = 1
## Returns a list: [sk, tk]
def getEucildean(int_a, int_b) :
    r0 = int_a; r1 = int_b; s0 = 1; s1 = 0; t0 = 0; t1 = 1; q = int(r0 /r1)
    while r1 != 0 :
        r_temp = r1; r1 = r0 -q * r1; r0 = r_temp
        s_temp = s1; s1 = s0 - q * s1; s0 = s_temp
        t_temp = t1; t1 = t0 - q * t1; t0 = t_temp
        if r1 != 0 :
            q = int(r0 / r1)
    return [s0, t0]

## Generated the valus of R_inverse and N_inverse based on value of R and N
## Used by MM block
```

```
## Returns [R_inverse, N_inverse]
def getRInverse_NInverse_for_MM (r, n) :
    temp = getEucildean(r, n)
    if ((temp[0] > 0)  and (temp[1] < 0)) :
        return [temp[0], -temp[1]]
    else :
        return [temp[0] + n, r - temp[1]]

 ## Generates P_inverse and Q_inverse for Barret Unit
 ## Returns [P_inverse, Q_inverse]
def getP_inv_Q_inv(p,q) :
    temp = getEucildean(p,q)
    num_pi = 0; num_qi = 0
    if temp[0] >= 0 :
        num_pi = temp[0]
    else :
        num_pi = temp[0] + q
    if temp[1] >= 0 :
        num_qi = temp[1]
    else :
        num_qi = temp[1] + p

    return [num_pi, num_qi]


## Return the random number of inputs and the results of outputs (kept in order)
def getRandom(inputs,outputs) :
    import random
    out = [None] * (len(inputs) + len(outputs))
    num_r = 2 ** 7

    ############ THE ORIGINAL INPUTS ################
    num_c = 6757 ##dummy value
    num_d = 6593 ##dummy value
    num_p = 103 ##dummy value
    num_q = 89 ##dummy value
    ################################################

    ############Generation of other inputs based on the inputs c, d, p, q which are
given by user#############################################
    num_pm1 = num_p - 1
    num_qm1 = num_q - 1
    num_pr = int((2**14) / num_p)
    num_qr = int((2**14) / num_q)
    num_pm1r = int((2**14) / num_pm1)
    num_qm1r = int((2**14) / num_qm1)
    num_pmm = getRInverse_NInverse_for_MM(num_r, num_p)[1]
    num_qmm = getRInverse_NInverse_for_MM(num_r, num_q)[1]
```

```
    (num_pi, num_qi) = getP_inv_Q_inv(num_p, num_q)
    num_rp = (num_r ** 2) % num_p
    num_rq = (num_r ** 2) % num_q
    num_cp = num_c % num_p
    num_cq = num_c % num_q
    num_dp = num_d % (num_pm1)
    num_dq = num_d % (num_qm1)
    num_mp = (num_cp ** num_dp) % num_p
    num_mq = (num_cq ** num_dq) % num_q

    if (num_mp - num_mq) >=0 :
        num_h = (num_qi * (num_mp - num_mq)) % num_p
    else :
        num_h = (num_qi * (num_mp - num_mq + num_p)) % num_p
    num_m = num_q * num_h + num_mq

    num_n = num_p * num_q
    num_check = num_p * num_pi + num_qi * num_q
    if num_check % num_n == 1 :
        print('Bazout identity satisfied!!')
    else :
        print('Bazout identity failed to check!')



##############################################################################################
################################################
    out[inputs.index('P')] = num_p
    out[inputs.index('Q')] = num_q
    out[inputs.index('P-1')] = num_pm1
    out[inputs.index('Q-1')] = num_qm1
    out[inputs.index('P_R')] = num_pr
    out[inputs.index('Q_R')] = num_qr
    out[inputs.index('P-1_R')] = num_pm1r
    out[inputs.index('Q-1_R')] = num_qm1r
    out[inputs.index('C')] = num_c
    out[inputs.index('D')] = num_d
    out[inputs.index('Q_I')] = num_qi
    out[inputs.index('P_MM')] = num_pmm
    out[inputs.index('Q_MM')] = num_qmm
    out[inputs.index('Rp')] = num_rp
    out[inputs.index('Rq')] = num_rq
    out[outputs.index('M') + len(inputs)] = num_m
    return out
```

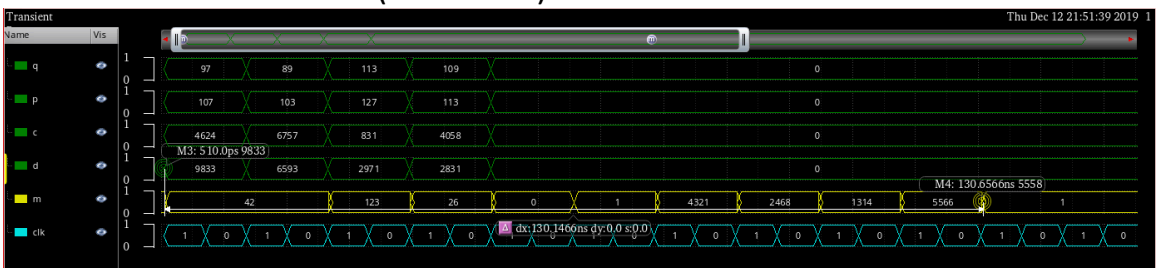
```
## Generates vector file uisng standard vector file format
def vec_generator (inputs = ['P','Q','P-1','Q-1','P_R','Q_R','P-1_R','Q-
1_R','C','D','Q_I','P_MM','Q_MM','Rp','Rq'], inputs_width
=[7,7,7,7,14,14,14,14,14,14,7,7,7,7,7], outputs = ['M'], outputs_width = [7],  number
= 30, clk = 20, delay = 2, unit = 'ns', slope = 0.01, vih = 1.8, vil = 0, file_path =
'/Users/yihaowang/Desktop', file_name = 'super_test_final_3.vec'):
    if (len(inputs) != len(inputs_width)) or (len(outputs_width) != len(outputs)) or
(number < 0) or (clk <= 0) or (delay < 0) or(slope < 0) :
        print('Parameter Error!')
    else :
        import os
        import random

        os.chdir(file_path)
        with open(file_name, 'w') as the_file, open(file_name.split('.')[0] +
'_golden.txt', 'w') as the_golden :

            print('radix', end = ' ', file = the_file)
            length = 0
            for i in inputs_width :
                length += i
            for i in range(length) :
                print('1', end = ' ', file = the_file)

            print('\nio', end = ' ', file = the_file)
            for i in range(length) :
                print('i', end = ' ', file = the_file)

            print('\nvname', end = ' ', file = the_file)
            for i in range(len(inputs)) :
                for j in range(inputs_width[i]) :
                    print(inputs[i] + '<' + str(inputs_width[i] - j - 1) + '>', end =
' ', file = the_file)
            print('\ntunit ' + unit , file = the_file)
            print('slope ' + str(slope), file = the_file)
            print('vih ' + str(vih), file = the_file)
            print('vil ' + str(vil), file = the_file)

            time = delay
            for i in range(number) :

                random_num = getRandom(inputs, outputs)

                print('\n' + str(time), end = '    ', file = the_golden)
                print('\n' + str(time), end = '    ', file = the_file)
                for j in range(len(inputs)) :
                    print(inputs[j] + '=' + str(random_num[j]), end = ' ', file =
the_golden)
```

```
                    print(dec_to_bin(random_num[j], inputs_width[j]), end = ' ', file
= the_file)
                for j in range(len(outputs)) :
                    print(outputs[j] + '=' + str(random_num[len(inputs) + j]), end =
'', file = the_golden)
                time += clk

            print('Successfully Gnererated!')


vec_generator(number = 1, clk = 20, delay = 5)

## For testing the program
#print(getRInverse_NInverse_for_MM(128,113))
#getRandom([],[])

end_time = time.time()
print('Run Time: %.6fs'%(end_time - start_time))
```

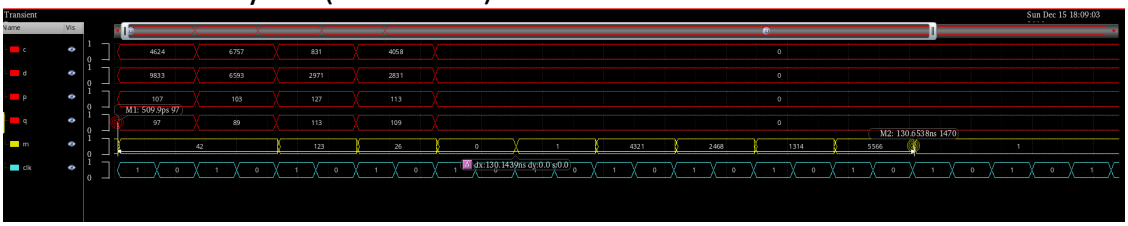
c. Function Test

Inputs								Outputs
Given				Python				Hardware
c	d	p	q	rp	p_i	rq	q_i	m
4624	9833	107	97	13	68	88	32	4321
6757	6593	103	89	7	70	8	22	2468
831	2971	127	113	1	105	112	9	1314
4058	2831	113	109	112	82	34	28	5566
Metric								
Time (ns)			Area (mm^2)			Area * Time		
130.65			0.851			111.24		

Waveform of Schematic (clk = 13ns):

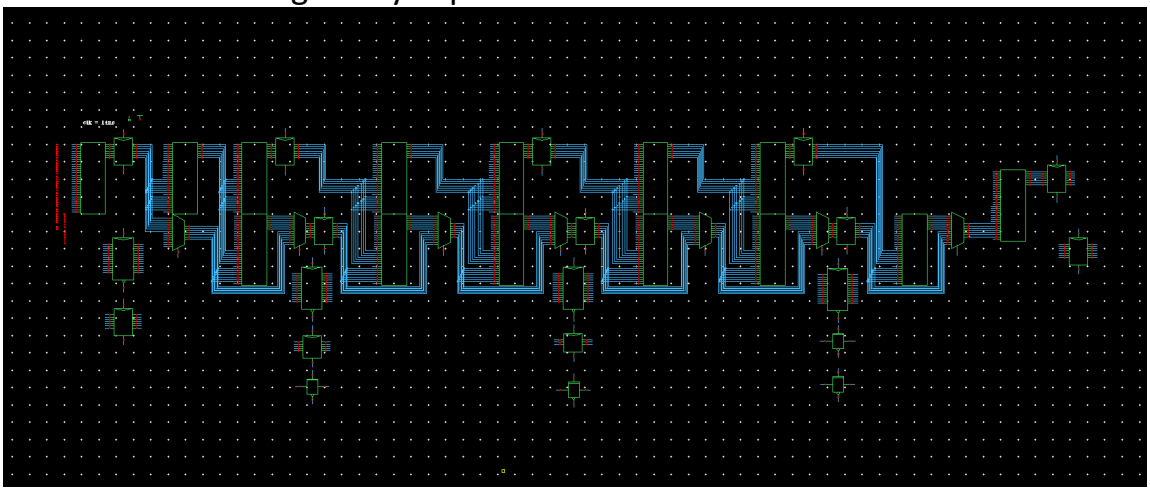


Waveform of Layout (clk = 13ns):

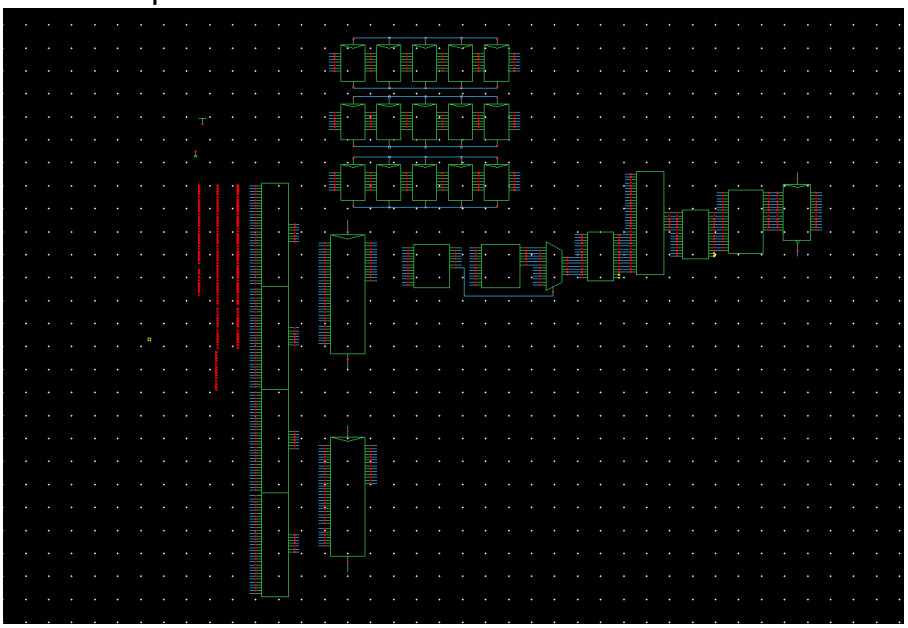


d. Screenshots

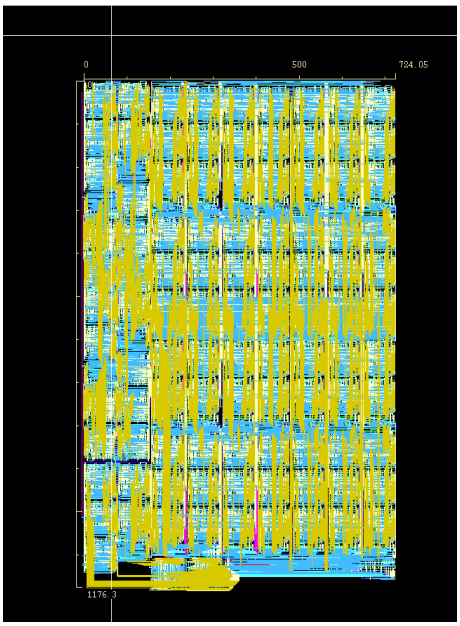
Schematic of Montgomery Exponential Unit:



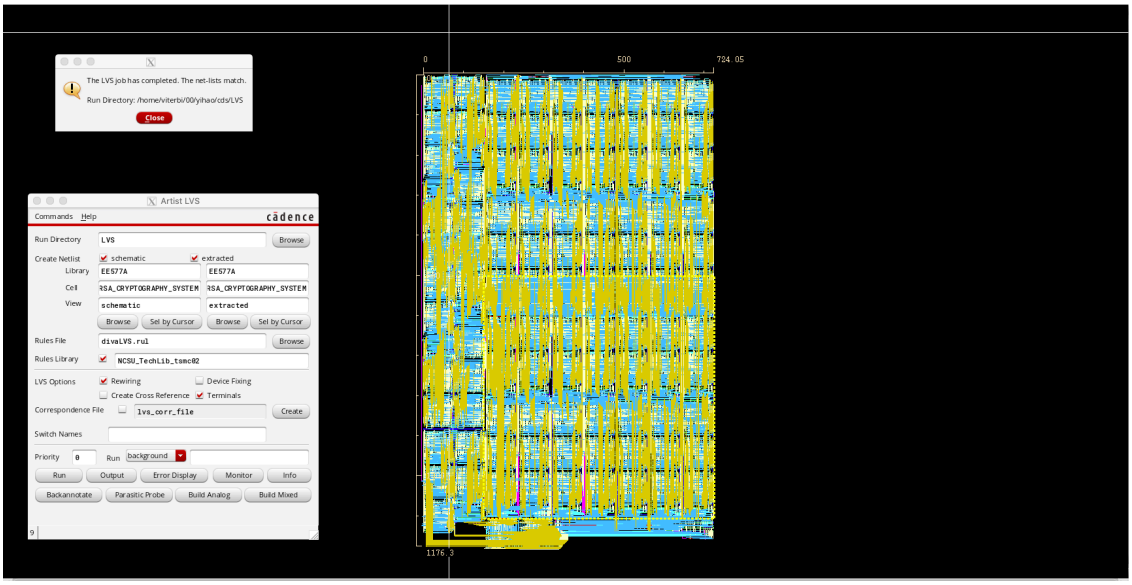
Schematic of Complete Circuits:



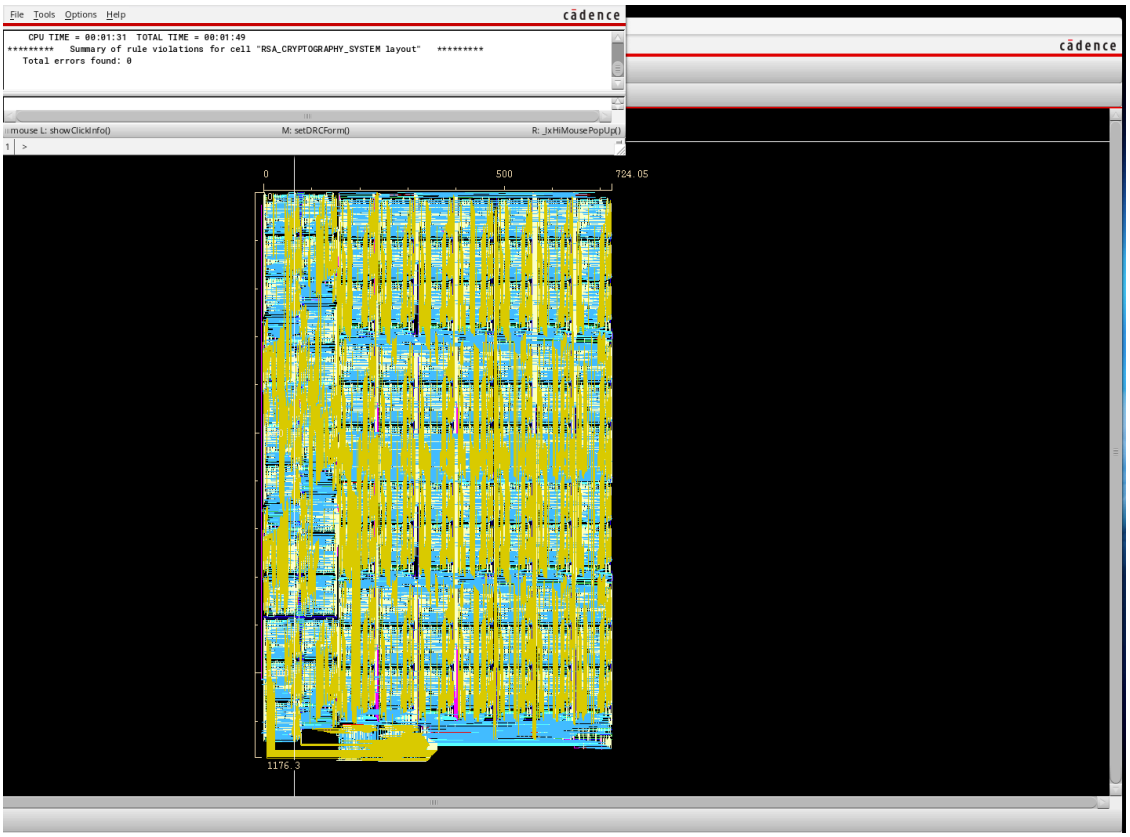
Layout:



LVS Match:



DRC:



e. Key Parameters

Height: 1176.3um
Width: 724.05um
Area: 851700 um^2
Minimum Clock: 13ns