PERANCANGAN DAN IMPLEMENTASI MODUL PERKALIAN MODULO MONTGOMERY UNTUK SISTEM KRIPTOGRAFI KUNCI PUBLIK RSA 512 *BIT* BERBASIS FPGA *(FIELD PROGRAMMABLE GATE ARRAY)*

Reza Irawan[1], M.Ary Murti[2], Koredianto Usman[3]
Department Teknik Elektro, Institut Teknologi Telkom
derez_flipmuth@yahoo.com[1], mam@ittelkom.ac.id[2], kru@ittelkom.ac.id[3]

ABSTRACT

One important point in data communication is security. This system is so important that users feel secured when communicating with other users. High-level security system can be applied for some data deliveries, such as e-mail, e-commerce, and others. Security system in data communication is well known as cryptography. One cryptography algorithm usually used for credit card coding is Rivest-Shamir-Adleman (RSA). RSA is an asymetric algorithm which uses different keys in its proccess. They are public key and private key. Public key is used for encryption and private key is used for decryption. Two main operations consisted in public key are modulo multiplication and exponential proccess. Modulo multiplication rules the RSA algorithm. An algorithm which can implement an effective multiplication is Montgomery. Montgomery can be applied in hardware because of its efficiency. 512-bit width is minimum value for RSA cryptography. The RSA construction starts from translating the multiplication algorithm into a computer. The design is drawn using VHDL and simulated using Modelsim SE 6.0. The implementation and synthesizing are proccessed by Xilinx ISE 8.1i. Finally, the result is transfered to a device named FPGA SPARTAN 3 XC3S1000 FT256-4C. This paper designs an implementation result using device target FPGA Spartan 3 XC3S1000 FT256-4C series showing top level entity which is capable and works on maximum frequency 41.625 MHz and requires slices 34% (2645 out of 7680), and 6% IOBs (12 out of 173).

Keywords: *Cryptography, RSA, FPGA, Montgomery.*

1. Pendahuluan

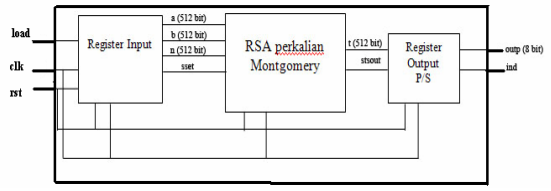
Tuntutan keamanan dalam pengiriman informasi/data berupa email, tanda tangan digital, dan kartu kredit pada sistem *e-commerce*. Otentikasi yang pada sistem e-commerce khususnya untuk kartu kredit mempunyai suatu standar khusus. Salah satu algoritma kriptografi yang banyak digunakan sekarang ini adalah *RSA* (*Rivest-Shamir-Adleman*). Algoritma *RSA* ini merupakan kriptografi asimetri dimana ada 2 buah kunci yang berbeda. Ada *public key* pada proses enkripsi dan *private key* pada proses deskripsi. Public key dapat diketahui oleh semua orang, tetapi private key hanya instansi yang mengetahuinya. Sehingga algoritma ini dirasa cukup aman untuk aplikasi *e-commerce*.

Kecepatan, komputasi daya, dan tingkat keamanan adalah hal-hal yang saling berpengaruh dalam kriptosistem ini. Adalah sesuatu yang sulit untuk mencapai kecepatan yang tinggi, komputasi yang rendah, dan tingkat keamanan yang tinggi dalam suatu rancangan. Karena pada umumnya yang tercapai adalah semakin tinggi tingkat keamanan yang diinginkan maka makin tinggi pula komputasi daya yang dibutuhkan dan makin rendah kecepatan yang dihasilkan.

Operasi perkalian modulo merupakan inti dari algoritma *RSA*. Operasi tersebut dimplementasikan oleh algoritma *Montgomery*. *Montgomery* merupakan algoritma paling efisien dan cocok untuk diimplementasikan pada perangkat keras. Lebar data 512 bit merupakan lebar data minimal untuk Kriptografi RSA.

Perancangan dimulai dengan menerjemahkan algoritma perkalian modulo tersebut ke dalam alur komputer. Rancangan ini dimodelkan dengan menggunakan bahasa pemrograman VHDL dan disimulasikan menggunakan Modelsim SE 6.0, kemudian disintesis dan diimplementasikan menggunakan Xilinx ISE 8.1i. Sedangkan devais target menggunakan board FPGA SPARTAN 3 seri XC3S1000 FT256-4C.

Blok dari sistem yang akan diimplementasikan adalah sebagai berikut:



Gambar 1. Blok Sistem

2. Algoritma Kriptografi RSA

Algoritma RSA ditemukan oleh R. L. Rivest, A. Shamir, dal L. Adleman pada tahun 1977[1]. RSA merupakan salah satu algoritma kunci publik yang menggunakan penyandian blok. Algoritma RSA dibuat berdasarkan fakta bahwa

perhitungan dengan komputer, menemukan bilangan prima besar sangat mudah, tetapi mencari faktor dari perkalian dua bilangan prima besar itu hampir tidak mungkin.

Algoritma utama sistem kriptografi RSA adalah sebagai berikut[1]:

a. Proses Enkripsi:
   $C = M^e \bmod n$
b. Proses Dekripsi:
   $M = C^d \bmod n$

Dengan, M : *plainteks*
C : *cipherteks*
e : kunci enkripsi.
d : kunci dekripsi.
n : modulo

Pada aplikasinya, bilangan *e dan n* diletakkan pada direktori publik pemilik sistem yang dapat dilihat oleh setiap orang yang ingin mengirimkan data kepada pemilik sistem.

Keamanan RSA berdasarkan pada besar bilangan yang digunakan sebagai kunci enkripsi dan kunci dekripsi, semakin besar bilangan yang digunakan maka kekuatan penyandian RSA juga semakin tinggi. Namun masalah yang dihadapi adalah kecepatan proses perhitungan juga semakin rendah dan kompleksitas yang semakin tinggi. Untuk itu pada setiap aplikasi dibuat optimasi yang tepat. Dalam sistem kriptografi RSA, salah satu kesempatan kriptanalisis untuk memecahkan sistem yaitu dengan memfaktorisasi bilangan *n.*

Namun dalam realisasi bilangan ini bergantung dari kegunaannya, masih banyak aplikasi saat ini yang menggunakan lebar bit yang lebih kecil. Untuk penyandian data yang menuntut keamanan yang sangat tinggi, digunakan lebar digit yang lebih besar, misalnya 512 bit atau 1024 bit[1].

2.1. Metoda Eksponensial *Montgomery*

Untuk melakukan komputasi sesuai dengan algoritma RSA ada dua bagian utama yang harus diperhatikan. Pertama Eksponensiasi Modular, kedua Perkalian Modular. Untuk bagian yang kedua akan diterangkan berikutnya. Dalam Eksponensiasi Modular, $C = M^e \bmod n$ tidak dihitung dengan memangkatkan M dengan e yang dilanjutkan dengan melakukan pembagian untuk memperoleh sisanya. Hal ini akan membutuhkan space yang sangat besar. Cara yang efisien adalah dengan mereduksi nilai sementara modulo n pada setiap langkah eksponensiasi. Dalam melakukan eksponensiasi harus diperhatikan berapa banyak perkalian modular untuk mencapai $C = M^e \bmod n$. Semakin sedikit operasi perkalian modular yang dilakukan maka hal itu semakin baik.

Cara yang paling kuno untuk menghitung $C = M^e \bmod n$, mungkin memulainya dengan menetapkan nilai awal $C = M \bmod n$ dan terus melakukan operasi perkalian modular $C = C.M \pmod n$, sampai mencapai $C = M^e \bmod n$. Pada metoda Eksponensial *Montgomery* dibutuhkan modul *Perkalian modulo Montgomery* (*MonPro*) yang akan dibahas secara detail pada subbab berikutnya. Algoritma Eksponensial *Montgomery* menggunakan metoda biner[3], seperti dapat dilihat berikut ini:

```
Function ModExp (M, n, e) , dengan n sebuah bilangan ganjil[3]
      Step 1. Hitung n' menggunakan algoritma Euclidean.
      Step 2. M̄ : = M . r mod n
      Step 3. x̄  := 1.r mod n
      Step 4. for i = k-1 downto to 0 do
      Step 5. x̄ : = Monpro (x̄, x̄)
      Step 6. If e_i = 1 then x̄ : = Monpro (M̄, x̄)
      Step 7. x : = Monpro (x̄, 1)
      Step 8  return x
```

Untuk mencari n' dan proses modul *Perkalian modulo Montgomery* (Monpro) dapat dilihat pada sub-bab dibawah ini. Dari algoritma di atas, dimulai dengan residu biasa M dan mendapatkan M.

2.2. Perkalian Modulo Algoritma *Montgomery*

Nilai $M^e \bmod n$ dihitung dengan mengimplementasikan fungsi perkalian modular pada bagian ini. Kita akan mempelajari algoritma untuk menghitung [3] : $R := (a \cdot b) \bmod n$, dimana *a, b, dan n* adalah *k*-bit. Karena *k* biasanya lebih dari 256, maka diperlukan suatu struktur data untuk menangani bilangan besar. Asumsikan *wordsize w* (w = 8, 16, 32), kita pecah *k*-bit menjadi *s word* hingga $(s-1)w < k \leq sw$. Pada perancangan digunakan k = 512 bit, w = 8 *bit* dan s = 8 *word*.

Pada tahun 1985 P.L *Montgomery* memperkenalkan algoritma yang lebih efisien untuk menghitung $R = a.b \bmod n$ dimana *a, b, n* adalah bilangan *k*-bit. Algoritma ini cocok digunakan untuk implementasi *software* dan *hardware* karena mempunyai kemampuan melakukan operasi aritmatik modulo pangkat 2 dengan cepat. Reduksi untuk menghasilkan *R* tidak dilakukan dengan cara pembagian, karena perhitungan yang dilakukan bukan modulo n tetapi modulo 2^s. Operasi ini mudah dilakukan karena representasi bilangan adalah dalam biner. Asumsikan modulus *n* adalah *k*-bit sedemikian hingga $2^{k-1} \le n < 2^k$. Perkalian modulo *Montgomery* didefinisikan sebagai berikut:

$$U = a \,.\, b \,.r^{-1} \bmod n \,,$$

dengan r^{-1} adalah invers dari r ($r = 2^k$; k = lebar n dalam biner $2^{k-1} \le n < 2^k$) yang mempunyai sifat berikut : $r^{-1} \,.\, r = 1 \bmod n$, dengan memanfaatkan sifat *n*-residu, maka untuk menghitung $R = a.b \bmod n$, diberikan input *a* atau *b* hasil dari *n*-residu misal *a* diganti dengan $\bar{a}$ sehingga:

$$\begin{aligned} U &= \bar{a} \,.\, b.\, r^{-1} \bmod n \\ &= (a.\, r \bmod n).\, b.\, r^{-1} \bmod n \\ &= a.b \bmod n \end{aligned}$$

Untuk menghitung perkalian modular *Montgomery* kita memerlukan n' dan r^{-1}. Cara memperolehnya yaitu dengan rumus

$r.\, r^{-1} - n\, n' = 1$ ($r = 2^k$; k = lebar n dalam biner $2^{k-1} \le n < 2^k$)

Lakukan modulo 2^w ke dua sisi maka kita dapatkan:

$r.\, r^{-1} \pmod{2^w} - n\, n' \pmod{2^w} = 1 \pmod{2^w}$

$- n\, n' \pmod{2^w} = 1 \pmod{2^w}$

Karena $- n\, n' \pmod{2^w} = - n_0 \, n_0'$; n_0 (least significant word) maka,

$- n_0 \, n_0' = 1 \pmod{2^w}$

$n_0' = - n_0^{-1} \pmod{2^w}$

$- no\, no' = a \;.\; 2^w + 1$ *(a : konstanta pengali)*

Karena n_0 ganjil maka FPB/*GCD* $(n_0, 2^w) = 1$, jadi pasti n_0' memiliki solusi. Secara lengkap algoritma perkalian modulo *Montgomery* [3] adalah sebagai berikut:

```
Function monpro(a,b)
Input : a,b,n ; output : U = a.b.r^-1 mod n ; r = 2^k(k =lebar n dalam biner ; 2^k-1<n<2^k)
      1. t : = a . b
      2. m : = t . n' mod r
      3. u : = (t + m . n)/r
      4. if u ≥ n then return u-n
                  else return u
```

Untuk perhitungan yang menggunakan input dengan lebar bit 512 bit maka dibutuhkan algoritma high-radix *Montgomery*[3] adalah pengembangan algoritma perkalian modular *Montgomery*. Algoritma ini pada dasarnya sama seperti di atas, tetapi ada perubahan di antaranya:

a. Proses perkalian modular dilakukan dengan memecah bilangan yang akan diolah ke dalam *wordsize w.* Sehingga bilangan integer yang besar tersebut dapat direpresentasikan dalam radix $W = 2^w$. Bilangan ini memerlukan s *word* dalam representasi radix *W*–nya, dengan mengambil $r = 2^{sw}$. Pada perancangan ini $W = 2^{512}$ dan $r = 2^{64 \times 8}$.
b. Perhitungan n' pada algoritma di atas relatif sulit dan lama, sedangkan dalam radix 2^w ini sangat sederhana karena yang dicari adalah n_0' (n_0 = least significant *word* dari *n*).
c. Karena operasinya dilakukan *word* demi *word* maka bisa mempercepat proses perhitungan

Perubahan Perkalian pada algoritma *Montgomery* [3] diatas adalah sebagai berikut

```
I. t := a.b
Dengan penjabaran lebih lanjut sebagai berikut:
          1.  For i = 0 to s-1
          2.   C := 0
          3.   For j = 0 to s-1
          4.   (C,S) := t i+j + aj. bi + C
          5.          t i+j := S
          6.          t i+s := C
II.  m := t . n' mod r
III. u := (t + m . n) / r
       Dengan penjabaran lebih lanjut sebagai berikut :
         7.  For i = 0 to s-1
         8.    C := 0
         9.    m := ti .no' mod 2^w
         10.   For j = 0 to s-1
         11.   (C,S) := t i+j + m . nj + C
         12.          t i+j  := S
         13.   For j = i + s to 2s-1
         14.          (C,S) := tj + C
         15.          tj := S
```

```
        16.   t 2s := C
IV.   if u ≥ n then return u-n else return u
      Dengan penjabaran sebagai berikut :
        17. For j = 0 to s
        18.   uj := tj+s
        19.  B = 0
        20.  For j = 0 to s
        21.   (B,D) : = uj – nj – B
        22.           vj : = D
        23. if B = 0 then return (vs-1vs-2 .....v0) else return (us-1us-2 .........u0)
```

3. Perancangan Sistem

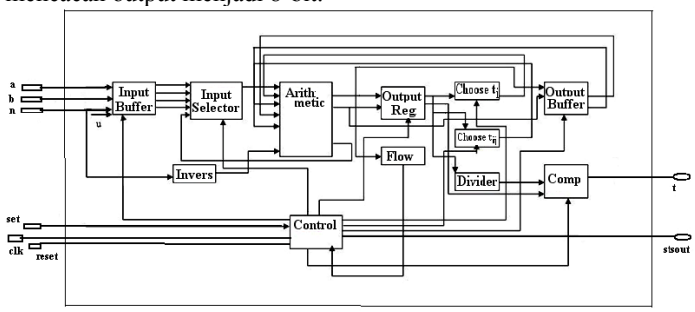
3.1 Metodologi Perancangan

Tahap perancangan dan implementasi algoritma kriptografi A5/2 ini menggunakan metode gabungan antara metode *Top-down* dan *Bottom-down.*

- *Top-down*
  Metode ini dilakukan dengan cara mendeskripsikan sistem secara keseluruhan beserta spesifikasi yang diinginkan dan membagi dalam modul-modul yang akan dirancang secara terpisah.
- *Bottom-up*
  Metode ini dilakukan dengan cara memilih modul-modul dasar, kemudian mengintegrasikan modul-modul tersebut menjadi suatu sistem yang lebih kompleks yang sesuai dengan spesifikasi perancangan.

3.2 Perancangan Arsitektur *RSA* Perkalian Modulo *Montgomery*

Pada makalah ini, perancangan *RSA* Perkalian modulo *Montgomery* ini menggunakan 11 buah modul dan 1 buah modul kontrol. Secara keseluruhan sebagai input adalah *a, b, n, clk, rst, dan set*. Keluarannya adalah stsout dan output. Input *a, b, n* masuk ke *Input Buffer Module* dan apabila set diberi *high* dan *clk* dalam keadaan aktif maka proses perkalian akan dimulai. Input *a, b,* dan *n* akan dipilih oleh input selector *word* demi *word* untuk menjalani perkalian *(a . b)*, input yang terpilih akan diproses oleh modul aritmatika. Operasi ini dikendalikan oleh controller. *Input Selector Module* digunakan untuk memilih data yang akan diproses. Proses selanjutnya adalah mencari *u*. Untuk mencari *u* diperlukan n_0' yang dihitung oleh *Invers Module*, data n_0' didapat dengan menggunakan algoritma yang dijelaskan pada bab 2. Proses yang terakhir yaitu proses mencari nilai *v (u - n)* sekaligus membandingkannya. Selektor output akan memilih apakah output yang harus keluar *u* atau *v*. Karena diagram blok tersebut mempunyai masukan dan output yang besar masing-masing 512 bit, maka hal ini menyebabkan modul di atas tidak bisa diimplementasikan dengan devais target FPGA. Karena keterbatasan I/O . Untuk itu maka ditambahkan modul tambahan yaitu register input sebagai penyimpan input & register output P/S yang berfungsi mencacah output menjadi 8-bit.



Gambar 2. *Main Module*

- ***Input Buffer Module***

Input Buffer Module ini bertugas untuk menahan sementara data input *a, b, n* dan *u* yang semuanya memiliki lebar 512 bit. Prinsip kerja dari *Input Buffer Module* ini adalah memilih sinyal 8 bit output yang berasal dari 512 bit yang kemudian diteruskan menjadi input bagi *Input Selector Module*. *Input Buffer Module* ini dikontrol oleh 4 buah sinyal selector yang berasal dari *Control Module*, yaitu sinyal *ssl1a, ssl1b, ssl2a, dan ssl2b*.

- ***Input Selector Module***

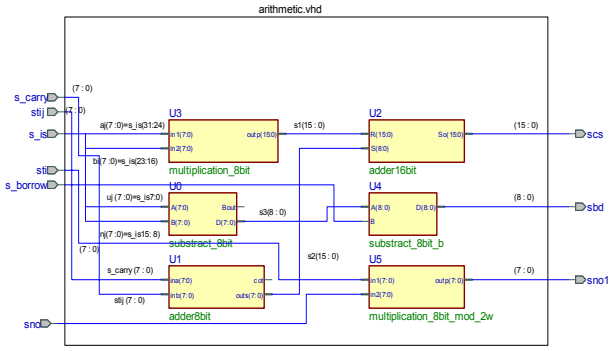
Input Selector Module ini digunakan untuk memiih data yang akan memasuki modul arithmetic

- ***Invers Module***

Invers Module ini bertujuan untuk mendapatkan nilai n_0'. Input dari modul ini adalah n_0 yang lebarnya 8 bit, n_0 adalah least significant word (w bit) dari modulus n. Output adalah sinyal n_0' (sno) yang memiliki lebar 8 bit. Output ini kemudian menjadi input bagi fungsi $m = t_i . n_0' . \mod 2^w$ yang terdapat pada *Arithmetic Module*.

- ***Arithmetic Module***

Modul ini befungsi untuk menghitung dan menjalankan fungsi-fungsi secara berurut sesuai dengan algoritma *RSA* perkalian *Montgomery*. Modul ini terlepas dari pengaruh control secara langsung karena input yang masuk ke dalam modul ini sudah diatur sedemikian rupa sesuai dengan langkah-langkah yang ada pada algoritma *RSA* perkalian *Montgomery*



Gambar 3. *Arithmetic Module*

- ***Output Register Module***

Modul ini berfungsi sebagai tempat pemberhentian sementara output dari *Arithmetic Module* dan kemudian disalurkan ke modul-modul selanjutnya, selain itu juga berfungsi menentukan letak bit-bit hasil proses sesuai dengan langkah-langkah yang telah ditempuh pada Perkalian *Montgomery.*

- ***Flow Module***

Modul ini digunakan sebagai tempat penyimpanan sementara dari bit ke 8 sinyal scs yang merupakan output dari *Arithmetic Module*

- ***Divider Module***

Modul ini digunakan untuk menentukan nilai *u*. Nilai *u* didapat dengan menggeser 512 bit output *s_d_cs* (1024 bit) ke kanan. Modul ini juga terlepas dari pengaruh control secara langsung karena hanya berfungsi untuk menggeser data yang masuk seperti yang dilakukan pada langkah 18

- ***Choose ti Module***

Modul Pemilih nilai ti ini digunakan untuk mendapatkan nilai ti yang dibutuhkan pada langkah 9.

- ***Choose tij Module***

Modul ini digunakan untuk mendapatkan nilai tij yang dibutuhkan pada langkah 1-18 Algoritma RSA perkalian *Montgomery*

- ***Output Buffer Module***

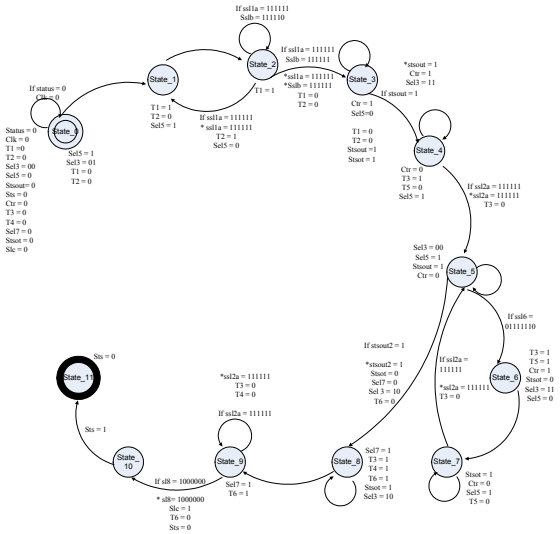
Modul ini digunakan untuk menghasilkan *C (carry)* dan *B (borrow)*.

- ***Comparator Module***

Comparator Module ini digunakan untuk menentukan output dari Algoritma Perkalian modulo *Montgomery* dengan cara menentukan apakah *u* atau *v* yang akan menjadi nilai output.

- ***Control Module***

Modul ini adalah modul yang mengatur urutan langkah-langkah dari modul–modul yang telah dijelaskan di atas sehingga tercapai urutan yang tepat dan sesuai dengan algoritma *RSA Perkalian modulo Montgomery*. Blok kontrol akan mengendalikan seluruh proses dari sistem yang telah didisain, bagian ini terdiri dari FSM *(Finite State Machine).* Diagram state di atas terdiri atas 11 state.



Gambar 4. FSM Dari Blok Control

4. Analisa Sistem

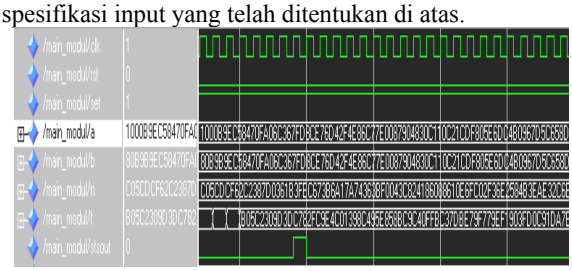
4.1 Analisa Hasil Perancangan (Simulasi)

Hasil perancangan dalam VHDL akan melalui tahap pengujian untuk memastikan bahwa sistem hasil rancangan bekerja sesuai dengan yang diharapkan, dalam hal ini apakah proses dari setiap modul dapat bekerja dengan. Dalam simulasi ini digunakan frekuensi clock sebesar 10 MHz. Berikut ini adalah hasil simulasi. Input yang digunakan untuk menguji rancangan yaitu *a, b*, dan *n*.

Tabel 1. *Input Main Module*

Input (512 bit)	Dalam *Hexadecimal*
A	1000B9EC58470FA06C367FD8CE76D42F4E86C 77E0087904830C110C21CDF805E6DC4B0967D 5C658D7AA434F34EDE9DC75FC8C6AFB69890F D0B1CA3407F1215E6
B	80B9B9EC58470FA06C367FD8CE76D42F4E86C 77E0087904830C110C21CDF805E6DC4B0967D 5C658D7AA434F34EDE9DC75FC8C6AFB69890F D0B1CA3407F1215E6
N	C05CDCF62C2387D0361B3FEC673B6A17A7436 3BF0043C824186088610E6FC02F36E2584B3E AE32C6BD521A79A76F4EE3AFE46357DB4C487 E858E51A03F890AF3

Berikut hasil *waveform*nya dari spesifikasi input yang telah ditentukan di atas.



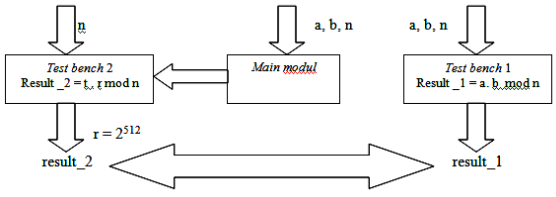
Gambar 5. *Waveform Main Module*

Dari hasil *waveform* dapat disimpulkan bahwa , saat *rst* bernilai 1 maka terjadi inisialisasi nilai awal "0" output dan sinyal-sinyal selector. Pada saat sinyal *set* ='1' maka clk, input, dan selector dari rancangan algoritma perkalian modulo *Montgomery* siap dimulai. Sinyal stsout bernilai 1 pada saat 1034400 ns hal ini menandakan bahwa keluaran akhir dari algoritma perkalian modulo *Montgomery* benar. Dari simulasi didapatkan nilai *output (t)* adalah sebagai berikut:

Tabel 2. *Output Main Module*

Output (t) 512 bit	B05C2309D3DC782FC9E4C01398C495 E858BC9C40FFBC37DBE79F779EF190 3FD0C91DA7B4C151CD3942ADE58658 90B11C501B9CA824B3B7817A71AE5F C076F50D

Hasil dari nilai output ini kemudian digunakan untuk melakukan verifikasi. Verifikasi rancangan dilakukan dengan menggunakan file VHDL.



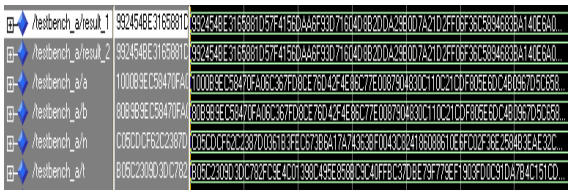
Gambar 6. Diagram Blok *Test Bench*

Pengujian dilakukan dengan membandingkan apakah result_1 = result_2 ? Apabila proses perhitungan hasil rancangan benar maka result_1 harus sama dengan result_2.

Result_1 = *a. b* mod *n* , dan

Result_2 = *t . r* mod *n*

(t merupakan keluaran dari rancangan main modul perkalian modulo *Montgomery* dimana t = *a. b.* r^{-1} mod *n* dengan *r* = 2^{512}). Dengan memanfaatkan *library numeric* yang ada di Modelsim SE plus 6.0, kita dapat melakukan verifikasi rangkaian dengan mudah. Dengan spesifikasi masukan a, b, n, dan t. Gambar dibawah ini memperlihatkan *waveform* dari *test bench* yang digunakan:



Gambar 7. *Waveform* Hasil Simulasi *Test Bench*

Dari hasil *waveform* di atas tampak bahwa result_1 = result_2 yang artinya rancangan telah memenuhi criteria sebagai modul pekalian modulo *Montgomery.*

4.2 Analisa Hasil Sintesis

Sebelum ke proses implementasi terlebih dahulu dilakukan proses sintesis. Proses sintesis dilakukan untuk merubah desain yang telah dibuat dalam bahasa VHDL menjadi suatu *gate level netlist.* Dalam *gate level netlist* ini terdapat keterangan mengenai interkoneksi antara level gerbang makro sel. *Software* yang digunakan untuk sintesis dan implementasi menggunakan Xilinx ISE 8.1i. Berikut ini cuplikan hasil sintesis dari sistem *top level entity.*

```
Selected Device : 3s1000ft256-4

 Number of Slices:                2645  out of   7680    34%
 Number of Slice Flip Flops:      1825  out of  15360    11%
 Number of 4 input LUTs:          4863  out of  15360    31%
 Number of bonded IOBs:             12  out of    173     6%
    IOB Flip Flops: 1
 Number of GCLKs:                    1  out of      8    12%

Timing Summary:
---------------
Speed Grade: -4
  Minimum period: 24.024ns (Maximum Frequency: 41.625MHz)
  Minimum input arrival time before clock: 22.542ns
  Maximum output required time after clock: 7.165ns
  Maximum combinational path delay: No path found
```

Dari laporan hasil sintesis terlihat bahwa sistem *top level entity* menggunakan *slices* 34% dari jumlah *slices* yang tersedia, *slices flip flop* 11*%,* 4 input LUTs 31%, bonded IOBs 6%, dan GCLKs 12%. Resources yang ada pada target devais ternyata jauh lebih besar dari resources yang dibutuhkan, sehingga desain ini dapat diimplementasikan pada target devais board FPGA Spartan 3 seri XC3S1000 FT 256-4C

4. 3 Analisa Hasil Implementasi

Setelah dilakukan proses sintesis maka proses selanjutnya adalah implementasi. Proses ini merupakan proses untuk memetakan rancangan menjadi gerbang-gerbang logika serta interkoneksinya dengan menggunakan *resources* yang tersedia pada FPGA yang menjadi target implementasi. Proses implementasi dibagi menjadi tiga bagian yaitu: proses *translate, map, dan place and route.*

4.3.1 Translate

Merupakan proses identifikasi file *netlist* yang dihasilkan dari proses sintesis lalu menggabungkan semua input *netlist* dan file *constraint.* Hasil proses *translate* adalah file *netlist* dalam format NGD yang siap dipetakan pada devais target.

4.3.2 Map

Pada proses ini dilakukan optimasi gerbang-gerbang logika dan jalur-jalur yang tidak terpakai pada netlist NGD dan memetakan rancangan tersebut ke dalam resources pada silicon. Pada proses ini dilakukan juga proses pemetaan elemen-elemen dasar menjadi sel-sel *Configurable Logic Blok* (CLB). Berikut ini cuplikan proses *map:*

```
Target Device  : xc3s1000
Target Package : ft256
Target Speed   : -4
Mapper Version : spartan3 -- $Revision: 1.34 $
Mapped Date    : Mon Aug 11 22:53:47 2008

Design Summary
--------------
Number of errors:      0
Number of warnings:    5
Logic Utilization:
  Total Number Slice Registers:     1,805 out of  15,360   11%
    Number used as Flip Flops:                1,797
    Number used as Latches:                       8
  Number of 4 input LUTs:           4,695 out of  15,360   30%
Logic Distribution:
  Number of occupied Slices:                       3,204 out of   7,680   41%
    Number of Slices containing only related logic:   3,204 out of   3,204  100%
    Number of Slices containing unrelated logic:          0 out of   3,204    0%
      *See NOTES below for an explanation of the effects of unrelated logic
Total Number 4 input LUTs:          4,729 out of  15,360   30%
  Number used as logic:             4,695
  Number used as a route-thru:         34
  Number of bonded IOBs:               13 out of     173    7%
    IOB Flip Flops:                    10
  Number of GCLKs:                      2 out of       8   25%

Total equivalent gate count for design:  47,025
Additional JTAG gate count for IOBs:  624
Peak Memory Usage:  174 MB
```

Dari ringkasan di atas dapat diketahui bahwa luas daerah yang digunakan untuk rancangan ini adalah 47.025 gates.

4.3.3 Place and Route

Place merupakan proses peletakan semua blok logika, sedangkan *Route* merupakan proses penentuan pin-pin pada blok logika. Pada proses ini akan menghitung *delay routing* yang terjadi antar blok FPGA. Selain itu *delay* pada komponen juga dihitung. Proses *place* and *route* akan menghasilkan file NCD yang siap dijadikan file BIT untuk proses *download* pada target *devices*. Berikut ini cuplikan *proses place and route:*

```
All signals are completely routed.

Total REAL time to PAR completion: 3 mins 12 secs
Total CPU time to PAR completion: 3 mins 10 secs

Peak Memory Usage:  201 MB

Placement: Completed - No errors found.
Routing: Completed - No errors found.

Number of error messages: 0
Number of warning messages: 4
Number of info messages: 1

Writing design to file main_modul_register.ncd


PAR done!
```

Setelah proses di atas selesai tanpa error, maka proses dapat dilanjutkan dengan proses *Generate* Programming File yang mengubah file NCD menjadi BIT yang siap untuk proses download pada target devais.

5. Kesimpulan

1. Hasil simulasi perancangan menunjukkan bahwa sistem *main module register* (*Top level Entity*) dapat me-*multiplex* 512 *bit ke* 8 *bit serial* .
2. Hasil simulasi perancangan sistem (*Main Module*) dengan Model sim 6.0 diperoleh jumlah clkcycle sebanyak 10345 untuk melakukan satu kali proses (512 *bit*) sehingga diperoleh troughput 0.049 bit/clkcycle.
3. Hasil Sintesis menggunakan Xilinx-ISE Project Navigator 8.1i menunjukkan bahwa maksimum frekuensi *clock* yang boleh digunakan untuk *top level entity* adalah sebesar 41.625 MHz
4. Slice yang dibutuhkan untuk implementasi dengan menggunakan Xilinx Spartan 3 XC3S1000 sebanyak 1825 *slices* dari 15360 *slices,* sedangkan IOB total yang digunakan adalah sebanyak 12 IOB dari 173 IOB.
5. Penggunaan *gates* sebanyak 47.025 gerbang.

Daftar Pustaka

[1] Kurniawan, Yusuf. (2004). *Kriptografi Keamanan Internet dan Jaringan Telekomunikasi.* Informatika, STTTelkom, Bandung.

[2] *RSA Algorithm.* DI Management Services, Sydney, Austarlia. (2006). http://www.di-mgt.com.au/rsa_alg.html, diakses terakhir tanggal 22 Oktober 2009.

[3] S. Kaliski JrBarkan, T. Acar. (1996). *Analyzing and Comparing Montgomery Multiplication Algorithms*. 16(3). IEEE Micro.

[4] J. Smith, Douglas. (1999). *HDL Chip Design*, Madison: Doone Publications.

[5] www.xilinx.com/products/silicon_solutions/fpgas/spartan_series/spartan3_fpgas/spartan3_product_table.pdf, diakses terakhir tanggal 22 Oktober 2009.

[6] CK, Koc. (1994). *RSA Hardware Implementation.* Version 2.0, Redwood City: RSA Laboratories, RSA Data Securitis Inc.

[7] R.L.Rivest, A.Shamir, L.Adleman. (1978). *A method for Obtaining Digital Signature and Publik-Key Cryptosystems.*

[8] O. Hwang, Enoch. (2004). *Microprocessor Design Principles and Practices With VHDL.* Brooks, Cole.

[9] L. Perry, Douglas. (2006). *VHDL Programming by Example.* Mc-Graw-Hill.