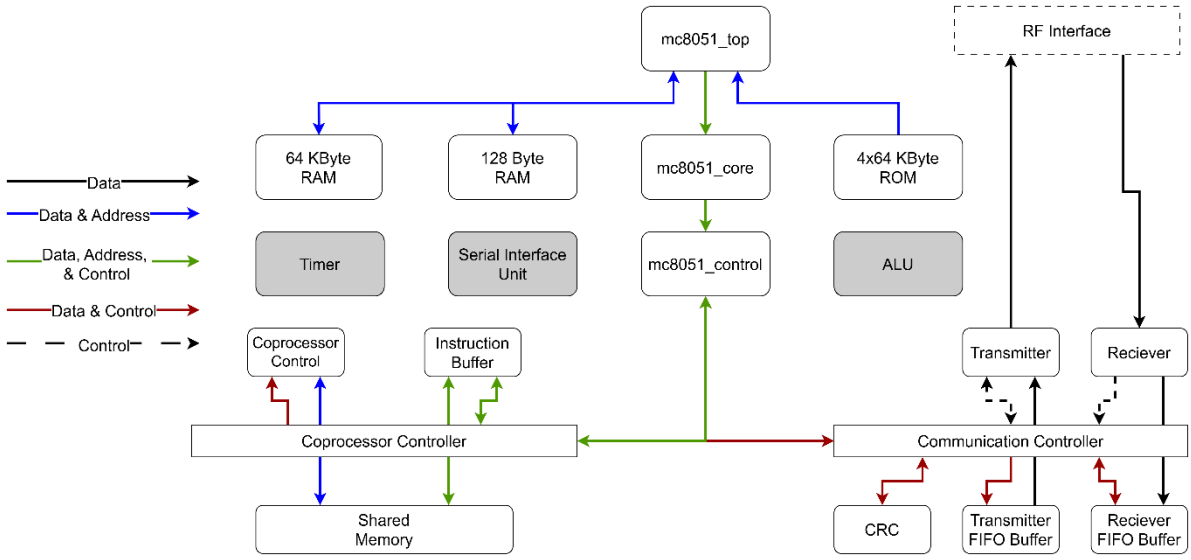
Desain Register Transfer Level (Ganang & Gotawa)

Desain

Koprosesor

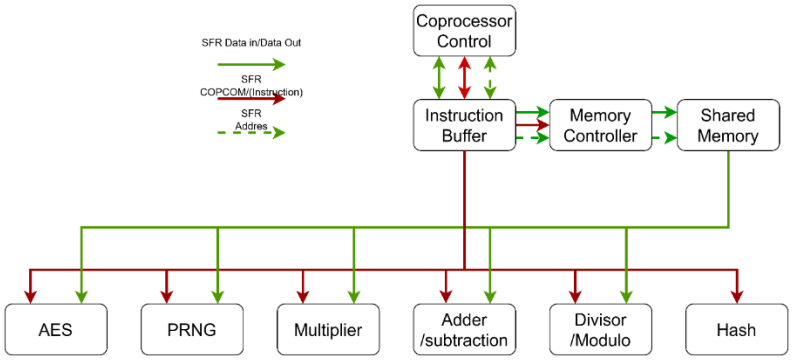
Gambar 1 menunjukkan arsitektur RTL sistem yang dirancang dengan mengintegrasikan co-prosesor akselerator dan koprosesor komunikasi dengan Mikrokontroler 8051 dari oregano system. Sistem ini terdiri dari beberapa komponen utama, yaitu mc8051 core, berbagai memori (RAM dan ROM), unit ALU, serta pengendali komunikasi dan koprosesor yang terhubung dengan antarmuka RF.



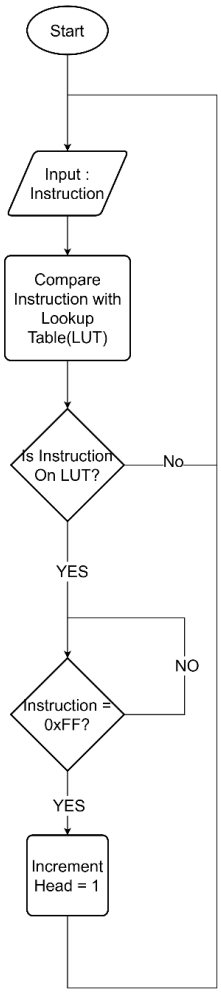
GAMBAR 28. DESKRIPSI UMUM ARSITEKTUR SMART CARD DENGAN MIKROKONTROLER 8051

Pada bagian inti, mc8051 core berinteraksi dengan berbagai blok memori seperti 64 KByte RAM, 128 Byte RAM, dan 4x64 KByte ROM melalui jalur data dan alamat. Arsitektur ini memiliki 4 buah ROM untuk mendukung fungsi code banking. Selain itu, terdapat modul Timer dan Serial Interface Unit yang merupakan arsitektur bawaan dari mikrokontroler 8051.

Selanjutnya, terdapat Coprocessor Controller yang berfungsi sebagai pengendali utama korposesor. Modul ini berkomunikasi dengan Shared Memory, Coprocessor Control, dan Instruction Buffer untuk mendukung eksekusi instruksi khusus yang memerlukan performa tinggi atau keamanan lebih, seperti kriptografi. Aliran data, alamat, dan sinyal kontrol diilustrasikan dengan warna berbeda sesuai dengan keterangan pada gambar. Sistem juga dilengkapi dengan Communication Controller yang menjadi jembatan antara mikrokontroler dan modul komunikasi eksternal. Di dalamnya terdapat unit CRC, Transmitter FIFO Buffer, dan Receiver FIFO Buffer. Komunikasi ke dunia luar dilakukan melalui Transmitter dan Receiver yang terhubung ke RF Interface.



GAMBAR 29. ARSITEKTUR COPROCESSOR CONTROLLER



GAMBAR 30. DIAGRAM ALIR PROSES PREFETCH PADA COPROCESSOR CONTROL.

Coprocessor Control

Gambar 2 menggambarkan arsitektur rinci dari Coprocessor Controller yang berfungsi sebagai

pengelola utama berbagai unit akselerator kriptografi dan aritmatika pada sistem smart card. Pada arsitektur ini, unit Coprocessor Control bertugas menerima dan mengatur instruksi yang diberikan melalui Special Function Register (SFR). Instruksi, data, serta alamat yang masuk akan disalurkan ke Instruction Buffer, yang kemudian bertanggung jawab menyimpan dan meneruskan instruksi ke unit-unit terkait di bawahnya. Selanjutnya, Instruction Buffer berinteraksi dengan Memory Controller untuk mengelola akses ke Shared Memory. Shared Memory ini digunakan sebagai media penyimpanan data sementara yang diperlukan dalam proses komputasi dan transfer data antar unit di dalam sistem.

Blok Coprocessor control adalah blok yang mengatur dan mengintegrasikan seluruh blok akselerator seperti AES, PRNG, Multiplier, Adder/subtraction, Divisor/Modulo, dan Hash. Selain itu, terdapat juga blok pengatur seperti Instruction Buffer dan Memory Controller. Terdapat pula blok yang berfungsi untuk mengintegrasikan atau menghubungkan port data dari masing-masing akselerator yaitu blok shared memory. terdapat beberapa proses pada blok ini yaitu :

- Prefetch
  Prefetch berfungsi untuk membaca instruksi dari mikrokontroler 8051 yang dikirimkan melalui SFR dan menyeleksi apakah instruksi tersebut valid atau tidak. Instruksi disebut valid jika instruksi tersebut dapat dieksekusi oleh akselerator atau memory controller. Proses ini bertujuan untuk mengurangi tumpukan data instruksi yang tidak dapat dieksekusi pada Instruction Buffer sehingga mencegah buffer tersebut cepat penuh. Proses prefetch berjalan seperti pada diagram alir 3 Instruksi yang masuk dibandingkan dengan Look-up Table (LUT). Jika instruksi yang masuk terdapat pada LUT, instruksi tersebut valid. Kemudian, proses menunggu sampai
  dengan masukan instruksi bernilai 0xFF. Lalu sinyal increment head dinaikan untuk memasukan instruksi ke Instruction Buffer.
- Fetch
  Proses fetch pada Coprocessor Control bertanggung jawab untuk mengambil data atau instruksi berikutnya dari Instruction Buffer agar dapat diproses oleh unit akselerator yang relevan. Ada dua hal yang diatur pada bagian ini yaitu :
  1. Increment Tail
     inyal s IncTail digunakan untuk mengatur pointer tail pada instruction buffer, yang menandakan data paling akhir yang telah di-fetch. Sinyal ini diperoleh
     dari hasil OR logika berbagai sumber, yaitu perintah increment tail dari setiap akselerator (misal: HASH, SHDMEM, AES, Adder, Multiplier, Divider, dan BC3). Dengan demikian, setiap akselerator dapat meminta data berikutnya dari buffer secara independen, dan proses fetch akan mengakomodasi permintaan tersebut.

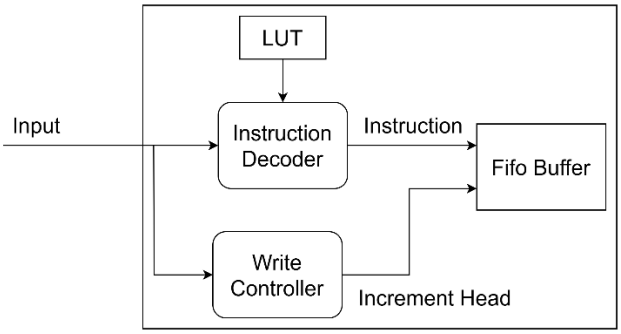
  2. Write Enable
     Sinyal s REBuffer mengatur apakah buffer diizinkan untuk dibaca atau
     tidak. Jika buffer tidak kosong (s EmptyBuffer = ’0’), maka s REBuffer akan diaktifkan (’1’), yang artinya data pada instruction buffer dapat diambil oleh akselerator. Jika buffer kosong (s EmptyBuffer = ’1’), maka s REBuffer dinonaktifkan (’0’) untuk mencegah pem-bacaan data yang tidak valid.

- Write back
  Proses writeback pada Coprocessor Control bertugas untuk menuliskan atau mengirimkan status terkini (Flag) dari masing-masing koprosesor serta kondisi buffer ke mikrokontroler 8051.

Instruction Buffer

Instruction Buffer (Ins Buffer) adalah komponen penting dalam arsitektur coprocessor yang

berfungsi sebagai penyangga (buffer) instruksi antara sistem utama dan unit eksekusi di dalam coprocessor. Instruction Buffer bekerja dengan prinsip FIFO. Untuk dapat bekerja, instruction buffer perlu mengatur pointer head dan pointer tail dengan dua buah kontrol unit. Arsitektur Instruction buffer dapat dilihat pada gambar 4.



GAMBAR 31. ARSITEKTUR INSTRUCTION BUFFER.

TABEL 5. DESKRIPSI PORT DAN DATA PADA INSTRUCTION BUFFER.

Input Port Name	Output Port Name	Data	Decription
i_Command	o_Command	Instruction	Instruksi untuk akselerator dan shared memory controller.
i_data	o_data	Addres Source	Alamat sumber untuk keperluan pembacaan data shared memory oleh mikrokontroler 8051 dan sebagai batas bawah.
i_AdTh	o_AdTh	Addres Threshold	Batas atas alamat penyalinan data secara blok.
i_AdDest	o_AdDest	Destination Address	Batas bawah alamat tujuan penyalinan data.

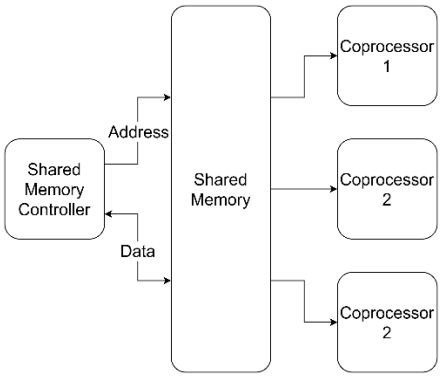
Shared memory Controller

Shared memory controller berfungsi untuk mengatur aliran data di shared memory seperti read, write, dan copy. Shared memory controller diatur oleh sinyal pada tabel 1. Berikut ini adalah fungsi yang bisa dijalankan oleh shared memory controller.

- Read : Membaca data pada shared memory dengan alamat Addres source.
- Write : Menuliskan data ke shared memory dengan alamat Destination Address.
- Copy : Melakukan penyalinan data dari alamat Adrress source ke alamat Destination Address.
- Block Copy : Melakukan penyalinan data pada blok tertentu di shared memory, data dari alamat Addres Source sampai dengan Address Threshold disalin ke alamat bawah Addres Destination.

Shared memory

Gambar 5 memperlihatkan hubungan antara shared memory, shared memory controller, serta beberapa coprocessor di dalam sistem. Setiap coprocessor terhubung ke shared memory untuk mengambil data input atau menuliskan hasil komputasi sesuai dengan alokasi yang telah ditentukan.



GAMBAR 32. ARSITEKTUR SHARED MEMORY.

Alokasi	Alamat bawah	panjang data(byte)	Alamat atas
in BC3 key	0	8	7
in BC3 data	8	8	15
out BC3	16	8	23
in AES key	24	32	55
in AES data	56	16	71
out AES Encrypt	72	16	87
in HASH	88	64	151
out HASH	152	32	183
in ADD A	184	32	215
in ADD B	216	32	247
out ADD	248	32	279
in MUL A	280	32	311
in MUL B	312	32	343
out MUL	344	64	407
in DIV A	408	32	439
in DIV B	440	32	471
out DIV Q	472	32	503
out DIV R	504	32	535
in XOR a	536	32	567
in XOR b	568	32	599
out XOR	600	32	631
out RNG	632	4	635
out AES Decrypt	636	16	651

Shared memory pada arsitektur ini berfungsi sebagai area pertukaran data utama antara coprocessor dan prosesor melalui memory controller. Setiap blok fungsional atau akselerato (seperti AES, HASH, ADDER, MULTIPLIER, DIVIDER, XOR, dan RNG) telah diberikan alokasi memori khusus sesuai dengan kebutuhan input dan output data, sebagaimana dirangkum pada Tabel 2. Coprocessor hanya diizinkan untuk membaca data pada segmen input (in) dan menulis pada segmen output (out). Sementara itu, Shared memory controller dapat membaca maupun menulis pada seluruh alamat di dalam shared memory.

Multiplier

Arsitektur multiplier yang ditunjukkan pada Gambar 6 dirancang untuk memproses dua buah input, yaitu A dan B, masing-masing dengan lebar data 256 bit. Untuk mengelola dan mengoptimalkan proses komputasi, data input ini dipecah ke dalam potongan-potongan yang lebih kecil melalui serangkaian selector (multiplexer).

Proses pemecahan data dimulai dengan membagi input 256 bit menjadi dua bagian 128 bit, kemudian masing-masing 128 bit dipecah lagi menjadi dua bagian 64 bit, lalu 64 bit dipecah menjadi dua bagian 32 bit. Proses pemecahan ini dilakukan secara bertahap menggunakan selector berdasarkan sinyal kontrol (Sel[5:0]), sehingga pada akhirnya didapatkan potongan data berukuran 32 bit yang siap untuk diproses lebih lanjut.

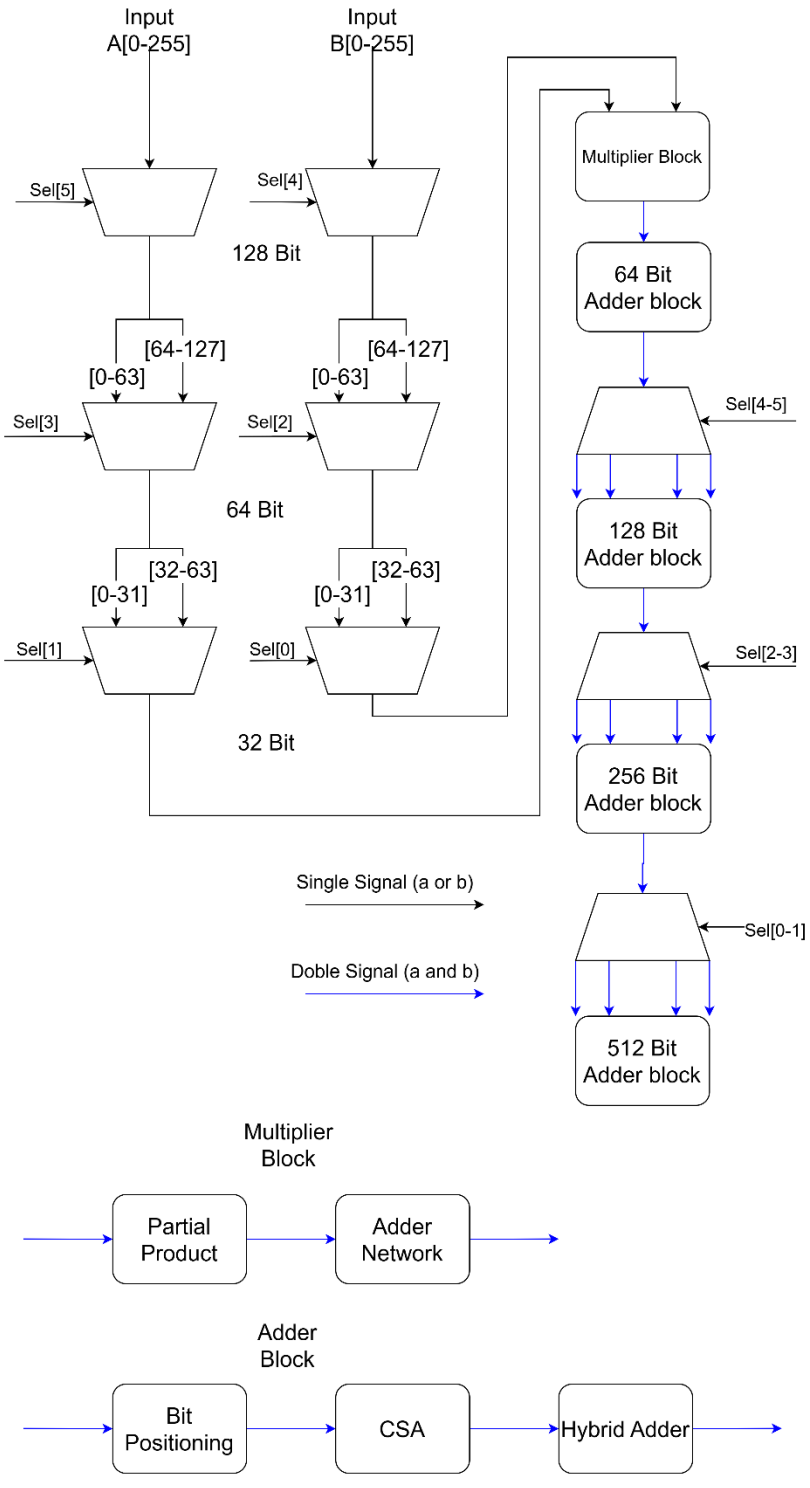
Setelah dipecah, data 32 bit dari kedua input (A dan B) dikirimkan ke Multiplier Block. Di dalam Multiplier Block terdapat dua tahapan utama:

- Partial Product Generator : Menghasilkan partial product dari pasangan input yang telah dipecah.
- Adder Network : Menjumlahkan partial product menggunakan jaringan penjumlah (adder network) secara efisien.

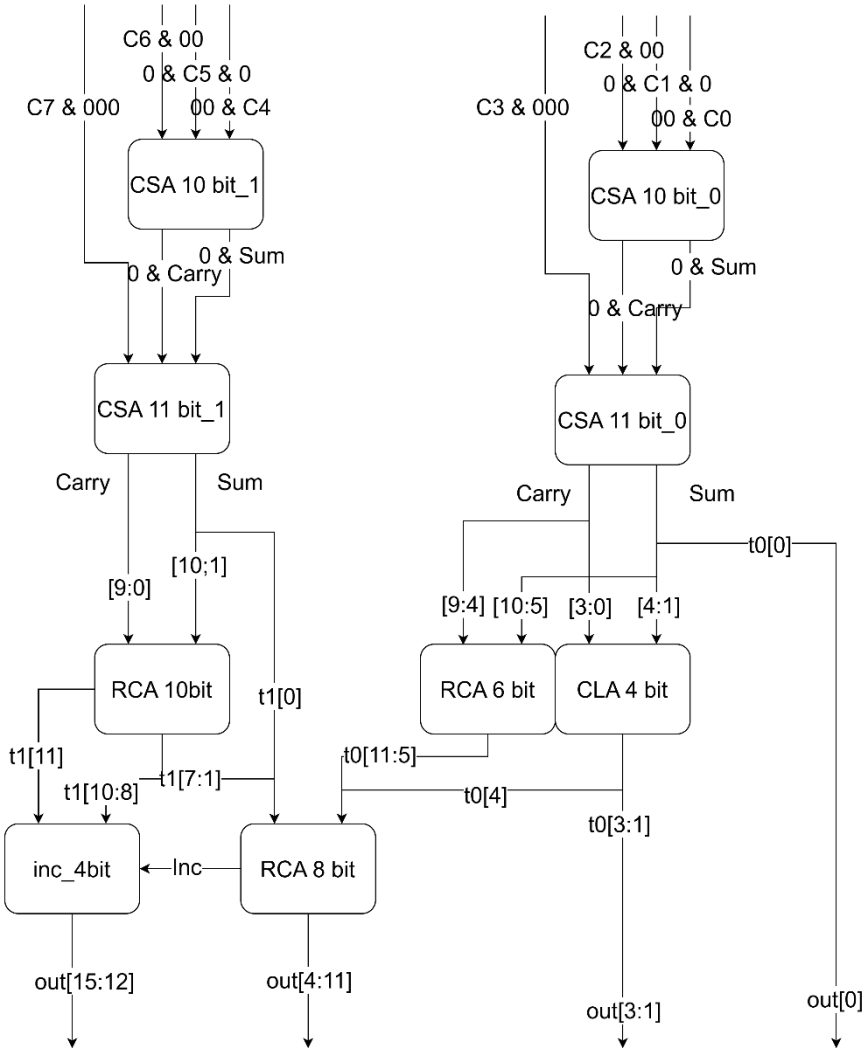
Hasil dari blok multiplier selanjutnya diproses oleh Adder Block. Adder Block terdiri dari tiga tahapan utama yaitu :

- Bit Positioning: Mengatur posisi setiap bit hasil perkalian agar penjumlahan berikutnya berada di posisi yang tepat.
- Hybrid Adder: Pada tahap akhir, penjumlahan diselesaikan menggunakan hybrid adder, yang terdiri dari rangkaian 16-bit Carry Lookahead Adder (CLA) yang disusun secara seri hingga mencapai panjang bit sesuai kebutuhan (misal: 64, 128, 256, hingga 512 bit).

Arsitektur adder block dirancang secara hierarkis dan modular. Setiap adder block dengan panjang bit tertentu (misal: 128 bit) menerima 4 masukan dari adder block berukuran setengahnya (64 bit). Mekanisme ini berlanjut secara rekursif hingga ke tingkatan paling rendah (32 bit). Dengan pendekatan seperti ini, proses penjumlahan dapat dilakukan secara paralel dan bertahap, sehingga meningkatkan efisiensi dan kecepatan komputasi pada operasi perkalian data yang sangat besar.



GAMBAR 33. ARSITEKTUR MULTIPLIER.



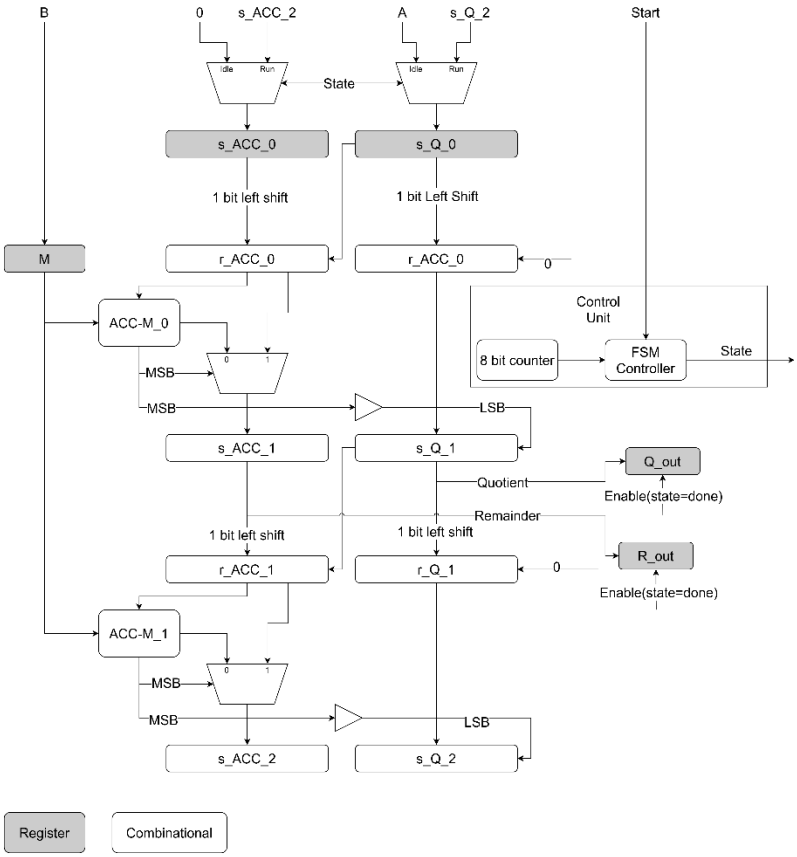
GAMBAR 34. ARSITEKTUR ADDER NETWORK

Divisor

Gambar 7 memperlihatkan arsitektur akselerator divisor yang diimplementasikan menggunakan algoritma non-restoring divider. Desain ini bertujuan untuk melakukan operasi pembagian dengan menghasilkan hasil bagi (Quotient) dan sisa pembagian (Remainder)

Pada diagram tersebut, terdapat dua masukan utama, yaitu A (dividend) dan B (divisor). Setiap proses pembagian terdiri dari beberapa tahapan yang dikendalikan oleh Control Unit yang berisi FSM (Finite State Machine) Controller dan 8 bit counter untuk mengatur iterasi langkah pembagian. Arsitektur divisor ini dapat melakukan dua langkah iterasi algoritma non-restoring divider dalam satu siklus clock. Hal tersebut dapat dilakukan dengan bantuan rangkaian kombinasional, kotak berwarna putih, dan register, kotak berwarna abu-abu. Terdapat register input yang akan mendapatkan input data dari selector. Selector tersebut menentukan input data berasal dari proses kombinasional iterasi

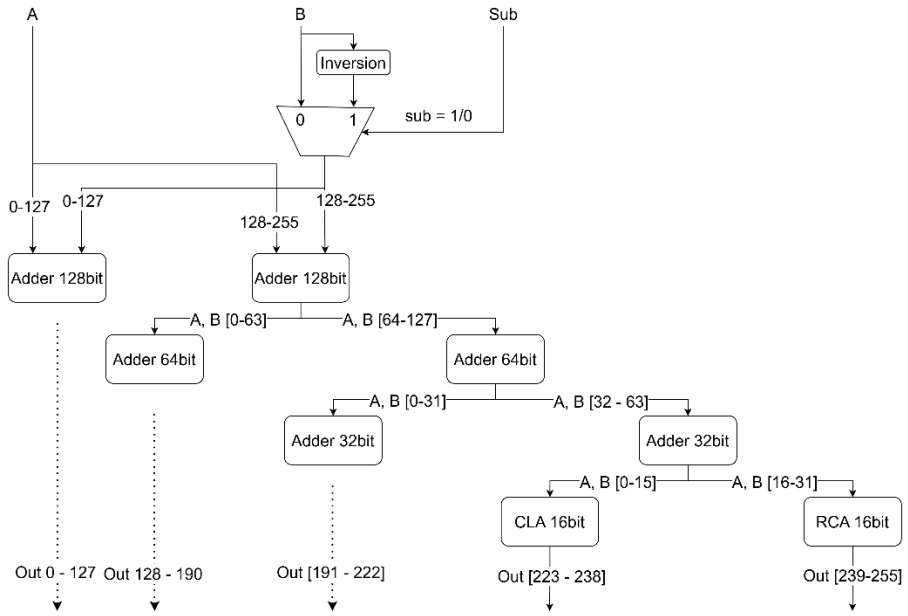
sebelumnya atau data baru. Kemudian, keluaran Quotient dan remainder disimpan pada register keluaran yang di-enable berdassarkan kondisi state.



GAMBAR 35. ARSITEKTUR DIVISOR

Adder/Subtractor

Gambar 8 memperlihatkan arsitektur akselerator yang mampu melakukan operasi penjumlahan (addition) dan pengurangan (subtraction) secara fleksibel hanya dengan mengatur sebuah flag input, yaitu Sub. Jika Sub bernilai ’0’, maka akselerator melakukan penjumlahan; sedangkan jika Sub bernilai ’1’, maka akselerator akan melakukan pengurangan dengan membalik (inversi) input B terlebih dahulu sebelum masuk ke tahap penjumlahan. Akselerator ini menerima dua input utama, yaitu A dan B, masing-masing dengan lebar data 256 bit. Data input ini diproses secara full kombinasional melalui hirarki struktur adder, mulai dari tingkat 128 bit, 64 bit, 32 bit, hingga pada tingkat paling bawah 16 bit. Proses ini dilakukan secara paralel, sehingga meningkatkan kecepatan operasi komputasi aritmatika.



GAMBAR 36. ARSITEKTUR ADDER DAN SUBTRACTOR.

Proses utama arsitektur ini meliputi:

- Input Selector dan Inversion: Input B dapat dilewatkan langsung atau terlebih dahulu melalui blok inversion tergantung nilai flag Sub. Selector akan memilih apakah B akan di-invers atau tidak.
- Struktur Hirarki Adder: Input A dan B dipecah menjadi dua bagian 128 bit dan diproses oleh Adder 128 bit. Selanjutnya, setiap bagian diproses secara bertingkat oleh Adder 64 bit, Adder 32 bit, dan pada tingkat paling bawah oleh CLA 16 bit atau RCA 16 bit.
- Parallel Output: Hasil dari masing-masing level adder digabungkan membentuk hasil akhir dengan lebar penuh 256 bit, dimana setiap blok adder mengisi rentang bit output yang spesifik.

Prosesor

Alamat SFR

Integrasi antara coprocessor dan mikrokontroler 8051 pada arsitektur ini dilakukan melalui pemanfaatan Special Function Register (SFR) yang terdapat pada ruang alamat ikrokontroler 8051. Tabel 3 memperlihatkan pemetaan alamat SFR yang digunakan dalam sistem ini.

TABEL 6. TABEL ALAMAT SFR.

Addres	0	1	2	3	4	5	6	7
F8	COPSTATR	COPMOSI	COPMISO	COPTH	COPSRC	COPDST		COPCOM
F0	B	COPWR	COPWREN	COPWRSTAT	COPRD	COPRDEN		COPRDSTAT

E8	COPSTATR2	COPMOSI2	COPMISO2	COPTH2	COPSRC2	COPDST2		COPCOM2
E0	ACC	COPWRLN	COPCRCINIT1	COPCRCO1		COPCRCEN		COPCRCI1
D8								
D0	PSW		COPCRCINIT2	COPCRCO2		COPCRCSTAT		COPCRCI2
C8								
C0	SCOUT							
B8	IP							
B0	P3							
A8	IE							
A0	P2							
98	SCON	SBUF						
90	P1							
88	TCON	TMOD	TLO	TL1	TH0	TH1		
80	P0	SP	DPL	DPH				PCON

TABEL 7. DESKRIPSI SFR

SFR	Description
COPWR	8-bit data from 8051 to FIFO transmitter
COPWREN	Write enable FIFO transmitter
COPWRSTAT	Transmitter status
COPRD	8-bit data from FIFO Transmitter to 8051
COPRDEN	Read enable fifo receiver
COPRDSTAT	Reciever status
COPCRCINIT1	Lower CRC initialization value (7 – 0)
COPCRC1	Lower CRC output (7 – 0)
COPCRCINIT2	Upper CRC initialization value (15 -8)
COPCRC2	Upper CRC output (15 – 8)
COPCRCEN	CRC start and data length selector
COPCRCSTAT	Status of CRC

TABEL 8. TABEL DESKRIPSI POSISI BIT SFR.

SFR	Bit Position							
	7	6	5	4	3	2	1	0
COPWRSTAT					Fifo full	Fifo empty	done	busy
COPRDSTAT					Fifo full	Fifo empty	done	busy
COPWR	Data							
COPRD	Data							
COPWREN								$\overline{En}$
COPRDEN								$\overline{En}$
COPCRCINIT1	Data							
COPCRCINIT2	Data							

COPCRCO1	Data							
COPCRCO2	Data							
COPCRCEN							Data length select	$\overline{En}$
COPCRCSTAT							done	busy

Memory

Code banking ROM

RAM

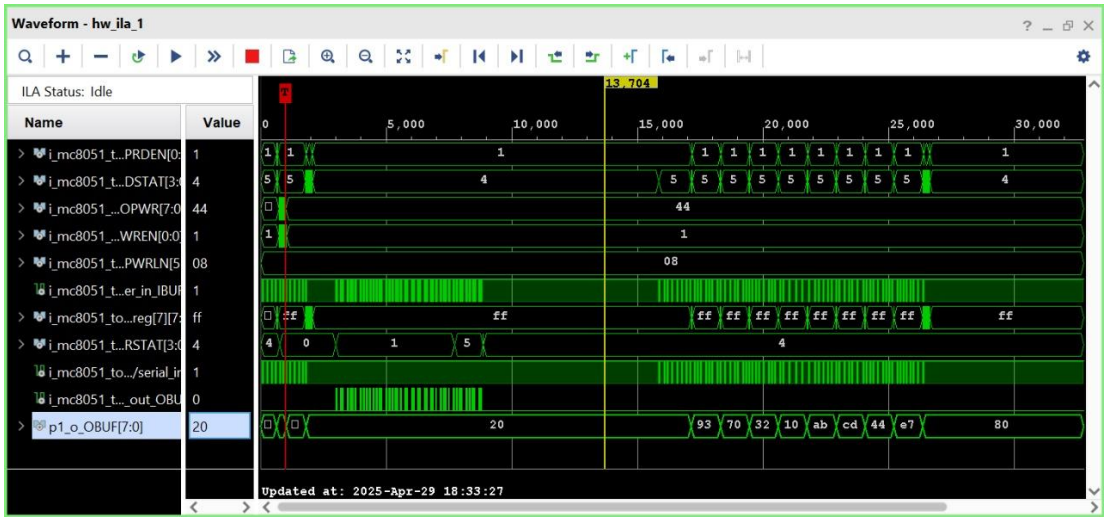
NVM

Implementasi

Pengujian

Koprosesor

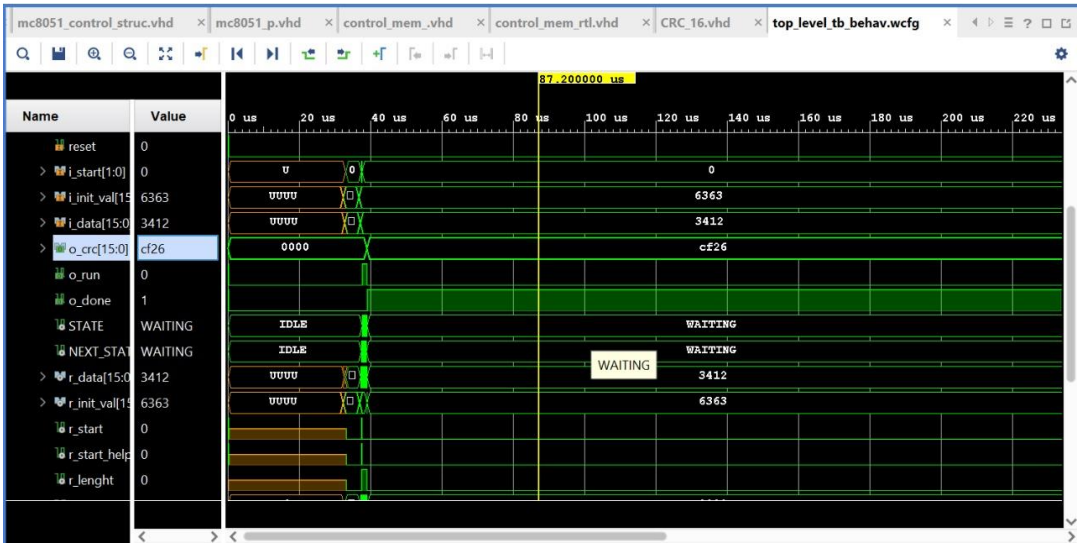
Anti-colision loop



GAMBAR 37. TAMPILAN ILA PENGUJIAN ANTI COLISION LOOP.

Variabel		

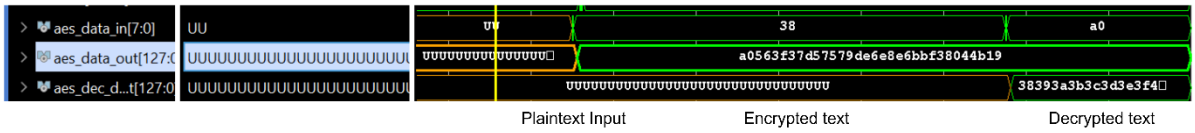
CRC-16



GAMBAR 38. HASIL SIMULASI PENGUJIAN CRC-16

Variabel/Prameter	Direction	Nilai
Initial	Input	0x6363
Data input	Input	0x12, 0x34
Data lenght selector	Input	‘1’
CRC out	Output	0xCF26

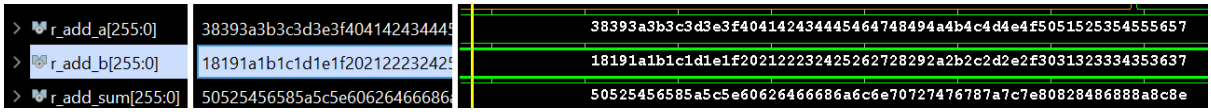
AES



GAMBAR 39. PENGUJIAN AES ENCRYPT DAN DECRYPT.

Variabel/Prameter	Direction	Nilai
Plaintext	In	0x38393a3b3c3d3e3f4041424344454647
Key	In	0x18191a1b1c1d1e1f202122232425262728292a2b2c2d2e2f3031323334353637
Encrypted text	Out	0xa0563f37d57579de6e8e6bbf38044b19
Decrypted Text	Out	0x38393a3b3c3d3e3f4041424344454647

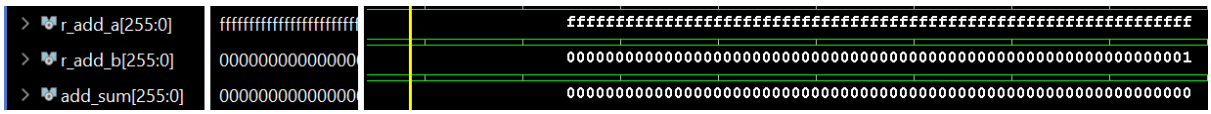
Addition



GAMBAR 40. PENGUJIAN ADDITION TEST VEKTOR 1.

GAMBAR 41. DATA PEMBANDING PENGUJIAN ADDITION TEST VEKTOR 1.

Variabel/Parameter	Direction	Nilai
A	In	0x38393A3B3C3D3E3F40414243444546474849 4A4B4C4D4E4F5051525354555657
b	In	0x18191A1B1C1D1E1F20212223242526272829 2A2B2C2D2E2F3031323334353637
sum	out	0x50525456585A5C5E60626466686A6C6E70727476787A7C7E8082 8486888A8C8E



GAMBAR 42. PENGUJIAN ADDITION TEST VEKTOR 2.

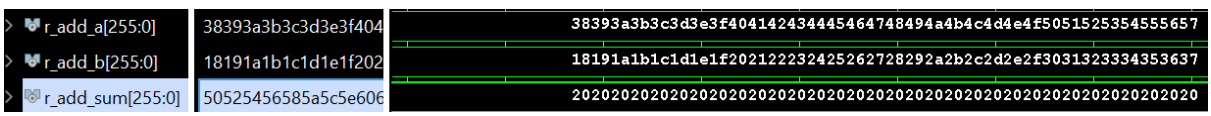
```
# Addition, test vector 2
# Define the hex numbers as Python integers
a = int("FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF", 16)
b = int("0000000000000000000000000000000000000000000000000000000000000001", 16)

# Compute the sum
s = a + b

# Print results
print("0x{:X} + 0x{:X} =".format(a, b))
print("    0x{:X}".format(s))
```

```
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF + 0x1 =
    0x100000000000000000000000000000000000000000000000000000000000000000
```

GAMBAR 43. PENGUJIAN ADDITION TEST VEKTOR 2.

Variabel/Parameter	Direction	Nilai
A	In	0x38393A3B3C3D3E3F40414243444546474849 4A4B4C4D4E4F5051525354555657
b	In	0x18191A1B1C1D1E1F20212223242526272829 2A2B2C2D2E2F3031323334353637
sum	out	0x50525456585A5C5E60626466686A6C6E70727476787A7C7E8082 8486888A8C8E

Subtraction



GAMBAR 44. PENGUJIAN OPERASI PENGURANGAN.

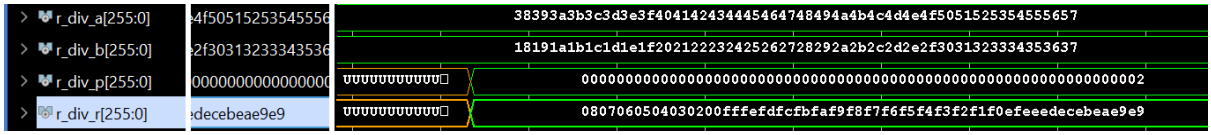
```
[1]: # Define the hex numbers as Python integers
a = int("38393a3b3c3d3e3f40414243444454647484944a4b4c4d4e4f5051525354555657", 16)
b = int("18191a1b1c1d1e1f202122232425262728292a2b2c2d2e2f3031323334353637", 16)

# Compute the sum
s = a - b

# Print results
print("0x{:X} + 0x{:X} =".format(a, b))
print("    0x{:X}".format(s))
```

```
0x38393A3B3C3D3E3F40414243444454647484944A4B4C4D4E4F5051525354555657 + 0x18191A1B1C1D1E1F202122232425262728292A2B2C2D2E2F3031
323334353637 =
    0x2020202020202020202020202020202020202020202020202020202020202020
```

GAMBAR 45. DATA PEMBANDING PENGUJIAN SUBTRACTION

Variabel/Parameter	Direction	Nilai
A	In	0x38393A3B3C3D3E3F40414243444454647484944A4B4C4D4E4F5051525354555657
b	In	0x18191A1B1C1D1E1F202122232425262728292A2B2C2D2E2F3031323334353637
sum	out	0x2020202020202020202020202020202020202020202020202020202020202020

Division



GAMBAR 46. PENGUJIAN DIVISION

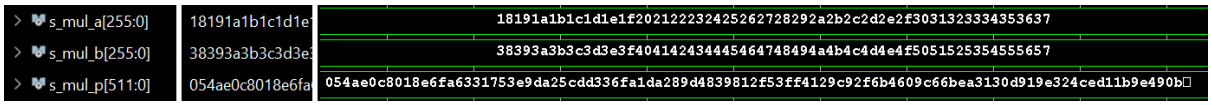
```
# Hex values
dividend_hex = "0x38393a3b3c3d3e3f40414243444454647484944a4b4c4d4e4f5051525354555657"
divisor_hex  = "0x18191a1b1c1d1e1f202122232425262728292a2b2c2d2e2f3031323334353637"

# Convert to integers
dividend = int(dividend_hex, 16)
divisor  = int(divisor_hex, 16)

# Define mask untuk 256-bit unsigned
mask = (1 << 256) - 1

# Terapkan mask (menghilangkan bit di luar 256-bit)
dividend_u = dividend & mask
divisor_u  = divisor  & mask

# Hitung quotient dan remainder secara unsigned
quotient, remainder = divmod(dividend_u, divisor_u)

# Tampilkan hasil
print("Quotient :", hex(quotient))
print("Remainder:", hex(remainder))
```

```
Quotient : 0x2
Remainder: 0x807060504030200fffefdfcfbfaf9f8f7f6f5f4f3f2f1f0efeeedecebeae9e9
```

GAMBAR 47. DATA PEMBANDING PENGUJIAN DIVISION.

Variabel/Parameter	Direction	Nilai
A	In	0x38393A3B3C3D3E3F40414243444454647484 94A4B4C4D4E4F5051525354555657
B	In	0x18191A1B1C1D1E1F202122232425262728292A2B2C2D2E2F3031323334353637
Quotient	out	0x0000000000000000000000000000000000000000000000000000000000000002
Remainder	out	0x0807060504030200fffefdfcfbfaf9f8f7f6f5f4f3f2f1f0efeeedecebeae9e9

Multiplier



GAMBAR 48. PENGUJIAN MULTIPLIER

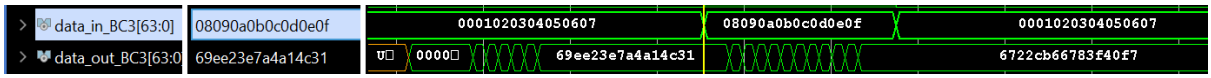
```
# Multiplier Test
# Define the hex numbers as Python integers
a = int("18191a1b1c1d1e1f202122232425262728292a2b2c2d2e2f3031323334353637", 16)
b = int("38393a3b3c3d3e3f404142434445464748494a4b4c4d4e4f5051525354555657", 16)
c = int("054ae0c8018e6fa6331753e9da25cdd336fa1da289d4839812f53ff4129c92f6b4609c66bea3130d919e324ced11b9e490bd69943c61011baf

# Compute the sum
s = a * b
check = s - c
# Print results
print("Hasil")
print("0x{:X} * 0x{:X} =".format(a, b))
print("    0x{:X}".format(s))
print("eror :")
print("    0x{:X}".format(check))
```

```
Hasil
0x18191A1B1C1D1E1F202122232425262728292A2B2C2D2E2F3031323334353637 * 0x38393A3B3C3D3E3F4041424344454647484 94A4B4C4D4E4F5051
525354555657 =
    0x54AE0C8018E6FA6331753E9DA25CDD336FA1DA289D4839812F53FF4129C92F6B4609C66BEA3130D919E324CED11B9E490BD69943C61011BAFBC40
3AAA8EE6B1
eror :
    0x0
```

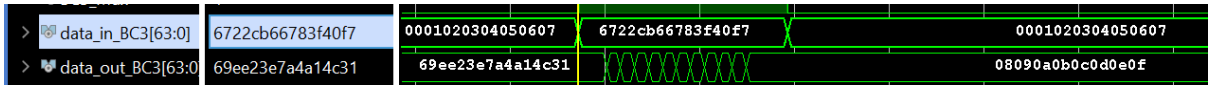
GAMBAR 49. DATA PEMBANDING MULTIPLIER.

Variabel/Parameter	Direction	Nilai
A	In	0x38393A3B3C3D3E3F40414243444546474849 4A4B4C4D4E4F5051525354555657
B	In	0x18191A1B1C1D1E1F202122232425262728292A2B2C2D2E2F3031323334353637
sum	out	0x054ae0c8018e6fa6331753e9da25cdd336fa1da289d4839812f53ff4129c92f6b4609c66bea3130d919e324ced11b9e490bd69943c61011bafbc403aaa8ee6b1

BC3



GAMBAR 50. PENGUJIAN ENKRIPSI DENGAN BC3.



GAMBAR 51. BC3 DEKRIPSI

Variabel/Prameter	Direction	Nilai
Plaintext	In	0x08090a0b0c0d0e0f
0xKey	In	0x0001020304050607
Encrypted text	Out	0x6722cb66783f40f7
Decrypted Text	Out	0x08090a0b0c0d0e0f