ANALISA DAN IMPLEMENTASI ENKRIPSI DATA TEXT DI EMBEDDED SYSTEM DENGAN ALGORITMA AES

SKRIPSI

Diajukan untuk Memenuhi Persyaratan

Memperoleh Gelar Sarjana Komputer

Disusun Oleh :

DHIMAS SAWUNG PAMUNGKAS

0910680082

PROGRAM STUDI TEKNIK INFORMATIKA

PROGRAM TEKNOLOGI INFORMASI DAN ILMU KOMPUTER

UNIVERSITAS BRAWIJAYA

2014

LEMBAR PERSETUJUAN

ANALISA DAN IMPLEMENTASI ENKRIPSI DATA TEXT DI EMBEDDED SYSTEM DENGAN ALGORITMA AES

SKRIPSI

KONSENTRASI KOMPUTASI BERBASIS JARINGAN

Untuk memenuhi sebagian persyaratan untuk mencapai gelar Sarjana Komputer

Disusun Oleh:

DHIMAS SAWUNG PAMUNGKAS

NIM. 0910680082

Skripsi ini telah diperiksa dan disetujui oleh dosen pembimbing pada tanggal 20 Desember 2013 :

Dosen Pembimbing I	Dosen Pembimbing II
Barlian Henryranu Prasetio, S.T, M.T.	**Eko Setiawan,ST.,M.Eng**
NIP. 821024 06 1 1 0254	**NIP. 870610 06 1 1 0256**

LEMBAR PENGESAHAN

ANALISA DAN IMPLEMENTASI ENKRIPSI DATA TEXT DI EMBEDDED SYSTEM DENGAN ALGORITMA AES

SKRIPSI

KONSENTRASI KOMPUTASI BERBASIS JARINGAN

Diajukan untuk memenuhi persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh :

DHIMAS SAWUNG PAMUNGKAS

NIM. 0910680082

Skripsi ini telah diuji dan dinyatakan lulus
pada tanggal 3 Januari 2014

Penguji I	Penguji II
Aryo Pinandito, ST, M.MT	**Gembong Edhi Setyawan, ST., MT.**
NIK.83051916110374	**NIK. 85092016110373**

Penguji III

Wijaya Kurniawan, ST., MT

NIP 820125 16 1 1 0291

Mengetahui
Ketua Program Studi Teknik Informatika/Ilmu Komputer

Drs. Marji, M.Si.

NIP. 19670801 199203 1 001

ORISINALITAS SKRIPSI

Saya menyatakan dengan sebenar-benarnya bahwa sepanjang pengetahuan saya, di dalam naskah SKRIPSI ini tidak terdapat karya ilmiah yang pernah diajukan oleh orang lain untuk memperoleh gelar akademik di suatu perguruan tinggi dan tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis dikutip dalam naskah ini dan disebutkan dalam sumber kutipan dan daftar pustaka.

Apabila ternyata didalam naskah SKRIPSI ini dapat dibuktikan terdapat unsur-unsur PLAGIASI, saya bersedia SKRIPSI ini digugurkan dan gelar akademik yang telah saya peroleh (SARJANA) dibatalkan, serta diproses sesuai dengan peraturan perundang-undangan yang berlaku. (UU No. 20 Tahun 2003, Pasal 25 ayat 2 dan Pasal 70).

Malang, 10 Januari 2014

Mahasiswa,

Dhimas Sawung Pamungkas

0910680082

KATA PENGANTAR

Puji Syukur kehadirat Tuhan Yang Maha Esa yang telah mencurahkan kasih dan rahmat, sehingga Proposal skripsi yang berjudul " ANALISA DAN IMPLEMENTASI ENKRIPSI DATA TEXT DI EMBEDDED SYSTEM DENGAN ALGORITMA AES " ini dapat diselesaikan.

Dalam menyelesaikan skripsi ini, penulis telah banyak mendapat bantuan dari berbagai pihak. Pada kesempatan kali ini, penulis mengucapkan banyak terima kasih kepada:

1. Allah S.W.T atas semua karunia-Nya, rahmat-Nya serta anugrah-Nya ini saya dapat menuliskan tugas akhir dengan baik.
2. Kedua orang tua penulis, Ayahanda Rudy Williyanto dan Ibunda Jumaani S.Pd yang selalu tidak lepas dari do'a dan harapan untuk terselesaikannya skripsi ini dan terus memberikan dorongan moral, material dan kasih sayangnya tiada akhir.
3. Kakak penulis Pricillia Ariesta dan Reno Nugroho serta Adik penulis Bayu Kristri Ambodo yang senantiasa memberi do'a dan motivasi. Tak lupa juga penulis berterima kasih untuk segenap keluarga besar penulis.
4. Bapak Drs. Marji, MT. dan Bapak Issa Arwani S.Kom, MSc. selaku Ketua Program dan Sekretaris Program Studi Teknik Informatika, segenap Bapak/Ibu Dosen dan seluruh Staff Teknik Informatika Universitas Brawijaya.
5. Bapak Barlian Henryranu Prasetio, S.T., M.T. selaku Dosen Pembimbing I yang telah banyak memberikan bimbingan, masukan dan arahan dalam penyusunan skripsi ini.
6. Bapak Eko Setiawan,ST., M.Eng. selaku Dosen Pembimbing II yang telah banyak memberikan bimbingan, masukan dan arahan dalam penyusunan skripsi ini.
7. Weni Prameswari, yang selalu menjadi motivasi penulis.
8. Teman penulis, Arga Suwastika S.Kom, Himawan Aditya P S.Kom., Thierry Rahman Aziz, S.Kom., Yusuf Oktofani S.Kom., Hoyi Ndadak Aji

S.Kom., Delis Sukmawati S.Kom, Darmawan Lahru R S.Kom., Yuri Citra P S.Kom. Angie Pramuditha A S.Kom., Alan Nur Abdan S.Kom., Nizamudin Ghonim S.Kom., M Khuril Ashari S.Kom., Hafidz Rozaq S.Kom., Wildan Ryan R S.Kom., Aldim Irfani V S.Kom, Riesky Ananda A, S.Kom yang selalu bertukar pikiran dan semangat selama pengerjaan skripsi ini serta senior penulis.

9. Mas Didit selaku Laboran Laboratorium siskombot yang selalu bersedia meminjamkan lab dan alat untuk melaksanakan percobaan dan pengujian.
10. Teman-teman Angkatan 2008, 2009, 2010, dan 2011 Teknik Informatika, terimakasih atas segala bantuannya selama menempuh studi di Program Teknologi Informasi dan Ilmu Universitas Brawijaya.
11. Pihak lain yang tidak bisa penulis sebutkan satu persatu yang terlibat langsung maupun tidak langsung demi terselesaikannya skripsi ini.

Hanya doa yang bisa penulis berikan dan semoga Allah SWT memberikan pahala serta balasan kebaikan yang berlipat. Amin.

Penulis menyadari bahwa Proposal skripsi ini jauh dari sempurna dan banyak kekurangan Untuk itu, saran dan kritik yang membangun sangat penulis harapkan. Semoga laporan skripsi ini membawa manfaat bagi penulis maupun pihak lain yang menggunakannya.

Malang, 2014

Penulis

ABSTRAKSI

Dhimas Sawung Pamungkas. 2013. Analisa dan Implementasi Enkripsi Data Text di Embedded System dengan Algoritma AES. Skripsi Program Studi Teknik Informatika. PTIIK Universitas Brawijaya. Barlian Henryranu Prasetio, S.T. , M.T. dan Eko Setiawan, ST.,M.Eng.

Embedded system telah mengalami perkembangan teknologi yang sangat pesat dan telah banyak dimanfaatkan. *Embedded system* dibangun untuk mengendalikan satu atau beberapa fungsi dan tidak dirancang untuk dapat diprogram oleh pengguna akhir seperti komputer pribadi. Pengguna banyak yang memilih beralih ke *embedded system* karena lebih effisien, murah, dan respon cepat, namun keamanan data di *embedded system* seringkali diabaikan. FPGA yang menggunakan bahasa pemrograman VHDL dan didukung dengan enkripsi AES merupakan solusi untuk menjaga keamanan data teks di *embedded system* tersebut. FPGA dibandingkan dengan *embedded system* lainnya misalnya Rasphberry ataupun Arduino, FPGA berjalan pada layer physical sehingga proses eksekusinya lebih cepat, sedangkan Raspberry ataupun Arduino berjalan pada layer aplikasi. Masalah keamanan data teks menggunakan enkripsi AES karena memiliki performa yang bagus, sulit untuk dipecahkan dan efisiensi waktu yang baik. Dalam skripsi " Analisa dan Implementasi Enkripsi Data Text di Embedded System dengan algoritma AES" dikaji dengan melakukan penelitian dan pembuatan prototipe sebuah sistem yang mempunyai validitas yang valid. Pengujian dilakukan dengan tiga tahap yaitu dengan pengujian fungsional system, validitas enkripsi, dan waktu enkripsi. Pengujian validitas dan fungsional memberikan hasil 100% system benar dalam menghasilkan enkripsi suatu data teks. Pengujian waktu memberikan hasil waktu yang diperlukan untuk mengenkripsi data teks di FPGA adalah 73.700000 us dan panjang karakter tidak mempengaruhi waktu yang digunakan untuk mengkripsi data teks selama panjang karakter tidak melebihi satu panjang blok yaitu 16 karakter atau 16 bytes.

Kata Kunci : *Validitas Enkripsi AES, FPGA, Embedded system, VHDL*

ABSTARCT

Dhimas Sawung Pamungkas. 2013. Analysis and Implementation of Encryption Data Text in Embedded System with AES Algorithm. Skripsi Informatics Technology / Computer Science Study. Information Technology and Computer Science Program Brawijaya University. Barlian Henryranu Prasetio, S.T. , M.T. dan Eko Setiawan, ST.,M.Eng.

Embedded system experiencing rapid technological development and has been used extensively. Embedded system is built for controlling one or some functions and not designed for programmable by last user like personal computer. Many user who choose move to embedded system because more efficient, low cost, and fast response, but data secure in embedded system often ignored. FPGA which use VHDL program language and supported with AES is a solution for keep security of data text in embedded system. FPGA compared with another embedded system for example Rasphberry or Arduino, FPGA work at physical layer so faster in execution process, whereas Rasphberry or Arduino work at application layer. Data text security using AES encryption because have good performance, hard to break, and good time efficiency. In essay "Analysis and Implementation of Encryption Data Text in Embedded System with AES Algorithm" studied with doing research and making prototype a system which have a validation. Examination will doing with tree step that are functional, validation and time examination. Validation and functional examination gave 100% true result in production encryption a data text. Time examination gave result of time of encryption data text in FPGA is 73.700000 us and character's long not affect time used for encryption data therefor character's long not exceed the block length is 16 characters or 16 bytes.

Keyword : *AES Encryption validation, FPGA, Embedded system, VHDL*

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB I
PENDAHULUAN

1.1 Latar Belakang

Manusia diciptakan sebagai makhluk sosial, oleh karena itu komunikasi sangat penting bagi antar manusia. Komunikasi terdapat dua jenis, yaitu komunikasi langsung dan komunikasi tidak langsung. Komunikasi langsung contohnya adalah manusia berbicara dengan manusia lain sedangkan komunikasi tidak langsung biasanya terjadi karena jaraknya yang jauh. Komunikasi tidak langsung terdapat berbagai cara missal, telpon, chating, email, webcam, social media dan sebagainya. Pada zaman yang super sibuk seperti ini banyak orang lebih memilih berkomunikasi dengan teks missal seperti aplikasi chat ataupun email, bahkan untuk mengirim perkataan penting manusia banyak yang melalui data teks karena untuk menghindari resiko untuk lupa. Data teks juga digunakan untuk mengamankan suatu akun dan system yaitu berupa password ataupun pin.

Aplikasi – aplikasi yang memerlukan data teks yang penting tersebut sekarang sudah ribuan jumlahnya, bahkan akhir – akhir ini banyak aplikasi-aplikasi yang beralih ke embedded system [Maman-2012]. Pengguna banyak yang memilih beralih ke embedded system karena lebih effisien, murah, dan respon cepat, namun keamanan data di embedded system seringkali diabaikan. Hal ini bukannya tanpa alasan. Embedded system adalah sistem kecil dengan resource dan processing power yang sangat terbatas [Maman-2012]. Penambahan implementasi security berarti memberikan tambahan kebutuhan akan dua hal tersebut, yang akhirnya mungking berakibat pada sistem yang menjadi semakin kompleks. Namun sekarang sudah ada embedded system yang mampu mengatasi kompleksitas sistem yaitu FPGA [Maman-2012]. FPGA dengan didampingi bahasa pemrograman VHDL mampu mengatasi kompleksitas sistem serta efisien [Maman-2012].

FPGA saja belum cukup untuk mengatasi keamanan data teks yang penting dalam embedded system, diperlukan lagi system untuk mengamankan data tersebut. Enkripsi lah yang akan menjadi pengaman data teks pada embedded system tersebut. Enkripsi yang memiliki performa yang bagus dan sulit untuk dipecahkan adalah AES [Sugi-2010]. FPGA yang memakai bahasa pemrograman VHDL dan dikombinasikan dengan algoritma AES akan mengatasi permasalah keamana data teks di embedded system.

Penelitian ini penulis mengambil judul skripsi Analisa dan Implementasi Enkripsi Data Text di Embedded System dengan algoritma AES. Algoritma AES digunakan untuk mengkripsi data text. Pengimplementasian enkripsi data text ini dikerjakan pada FPGA. Diharapkan pada penelitian ini menjadi langkah awal diciptakannya enkripsi data text berbasis *embedded system*.

1.2 Rumusan Masalah

Berdasarkan uraian latar belakang di atas, maka dapat dirumuskan permasalah pada skripsi ini yaitu sebagai berikut:

1. Bagaimana perancangan Enkripsi data text dengan algoritma AES pada FPGA Spartan 3e
2. Bagaimana pengimplementasian Enkripsi data text dengan algoritma AES pada FPGA Spartan 3e
3. Bagaimana pengujian fungsional system,validitas dan waktu Enkripsi data text dengan algoritma AES pada FPGA Spartan 3e

1.3 Batasan Masalah

Agar permasalahan yang dirumuskan dapat lebih terfokus, maka pada penelitian ini dibatasi oleh hal-hal berikut :

1. Enkripsi data text di implementasikan di papan FPGA Spartan 3e.
2. Penelitian ini hanya dititik beratkan pada proses Enkripsi.
3. Panjang karakter yang dipakai 16 karakter untuk plain text dan 32 karakter yang ditampilkan pada cipher text dalam bentuk HEX.
4. Algoritma enkripsi yang digunakan adalah algoritma AES 128 bit.

1.4 Tujuan

Penulisan skripsi ini mengimplementasikan enkripsi data teks dengan algoritma AES pada papan FPGA Spartan 3e untuk menjaga keamanan data teks pada level hardware.

1.5 Manfaat

Penulisan skripsi ini diharapkan mempunyai manfaat yang baik dan berguna bagi pembaca dan penulis. Adapun manfaat yang diharapkan adalah sebagai berikut:

A. Bagi penulis

1. Sebagai media untuk pengimplementasian ilmu pengetahuan teknologi yang telah didapat selama mengikuti perkuliahan di Teknik Informatika Universitas Brawijaya.
2. Penulis mendapatkan pengetahuan dan wawasan terkait kriptografi dan *embedded system* yang dalam penelitian ini menggunakan FPGA.

B. Bagi pembaca

1. Pembaca mendapatkan wawasan akan pengimplementasian program pada FPGA.
2. Sebagai bahan dan acuan dalam mengembangkan dan melanjutkan riset yang memiliki keterkaitan dengan penelitian ini.

1.6 Sistematika Penulisan

Sistematika penulisan dalam skripsi ini sebagai berikut:

BAB I Pendahuluan

Bab I Pendahuluan menguraikan latar belakang skripsi, rumusan masalah tentang pengembangan penelitan, tujuan dan manfaat skripsi serta sistematika penyusunan laporan skripsi.

BAB II Tinjauan Pustaka

Bab II menguraikan teori-teori yang menjadi referensi dalam pelaksanaan skripsi.

BAB III Metode Penelitian dan Perancangan

Bab III menguraikan tentang metode dan langkah kerja yang dilakukan dalam proses perancangan dan implementasi pada pelaksanaan skripsi dan perancangan sistem yang menjadi objek studi kasus skripsi.

BAB IV Implementasi

Bab IV menguraikan proses implementasi dari dasar teori yang telah dipelajari sesuai analisis dan perancangan sistem.

BAB V Pengujian dan Analisis

Bab V memuat hasil pengujian dan analisis terhadap sistem yang telah direalisasikan.

BAB VI Penutup

Bab VI memuat kesimpulan yang diperoleh dari pembuatan dan pengujian program, serta saran-saran untuk pengembangan lebih lanjut.

BAB II
TINJAUAN PUSTAKA DAN DASAR TEORI

Bab ini berisi tinjauan pustaka yang meliputi kajian pustaka dan dasar teori yang diperlukan untuk penelitian. Kajian pustaka membahas penelitian yang telah ada dan yang diusulkan. Dasar teori membahas teori yang diperlukan untuk menyusun penelitian yang diusulkan.

2.1 Kajian Pustaka

Kajian pustaka membahas penelitian yang telah ada dan yang diusulkan. Pada penelitian ini kajian pustaka diambil dari beberapa penelitian yang relevan yang pernah dilakukan. Penelitian yang relevan akan dibahas pada sub bab selanjutnya.

2.1.1 Analisis Algoritme dan Waktu Enkripsi Versus Dekripsi Pada Advanced Encryption Standard (AES) oleh Sugi Guritman, Ahmad Ridha dan Endang Purnama Giri

Riset tentang efisiensi waktu AES dalam mengamankan data telah dilakukan oleh Sugi Guritman dan kawan kawan. Riset ini berisi tentang pengujian kecepatan AES. Kesimpulan dari riset ini adalah Proses enkripsi AES didesain untuk melakukan penyandian secara rahasia dengan tingkat keamanan tak linear dengan kompleksitas waktu seefisien mungkin melalui penggunaan proses-proses transformasi yang ringan dalam implementasi. Hasil dari riset ini mengenai uji validitas dan waktu yang dilakukan dengan cara T-test dapat dilihat pada gambar 2.1

Tabel 2. Tingkat kinerja waktu enkripsi dan dekripsi berbagai *platform* (Nechvatal *et al.* 2000)

	MARS	RC6	AES (Rijndael)	Serpent	Two fish
32 bit (C)	II	I	II	III	II
32 bit (Java)	II	I	II	III	III
64 bit (C and Assembler	II	II	I	III	I
8 bit (C and Assembler	II	II	I	III	II
32 bit smartcard	II	I	I	III	III
Digital Signal Processor	II	II	I	III	I

Tabel 3. Tingkat kinerja waktu penjadwalan *key* berbagai *platform* (Nechvatal *et al.* 2000)

	MARS	RC6	AES (Rijndael)	Serpent	Two fish
32 bit (C)	II	II	I	III	III
32 bit (Java)	II	II	I	II	III
64 bit (C and Assembler	III	II	I	II	III
8 bit (C and Assembler	II	III	I	III	II
Digital Signal Processor	II	II	I	I	III

Tabel 4. Tingkat kinerja waktu keseluruhan (Nechvatal *et al.* 2000)

	MARS	RC6	AES (Rijndael)	Serpent	Two fish
Encl Dec	II	I	I	III	II
Key Setup	II	II	I	II	III

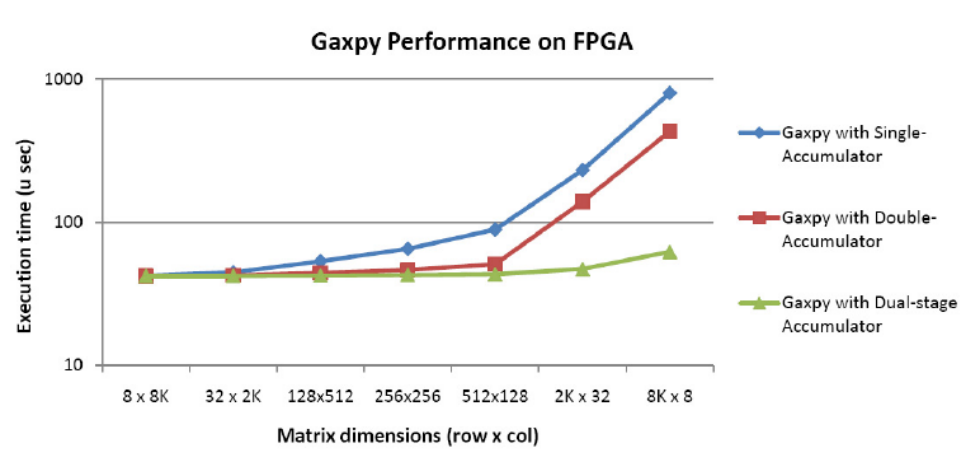
Gambar 2. 1 Hasil dari Riset
Sumber : [Sugi-2010]

Pada Gambar 2.1 tersebut menyatakan level kinerja waktu tinggi, II menyatakan level kinerja waktu rata-rata, dan III digunakan untuk menyatakan level kinerja waktu rendah. Dari data didapatkan algoritma kriptografi AES memiliki kinerja waktu tinggi yang konsisten untuk seluruh proses meskipun akan mengalami kinerja bagi blok kunci 192 bit dan 256 bit.

2.1.2 BLAS Comparison on FPGA, CPU and GPU oleh Srinidhi Kestur, John D. Davis, Oliver Williams

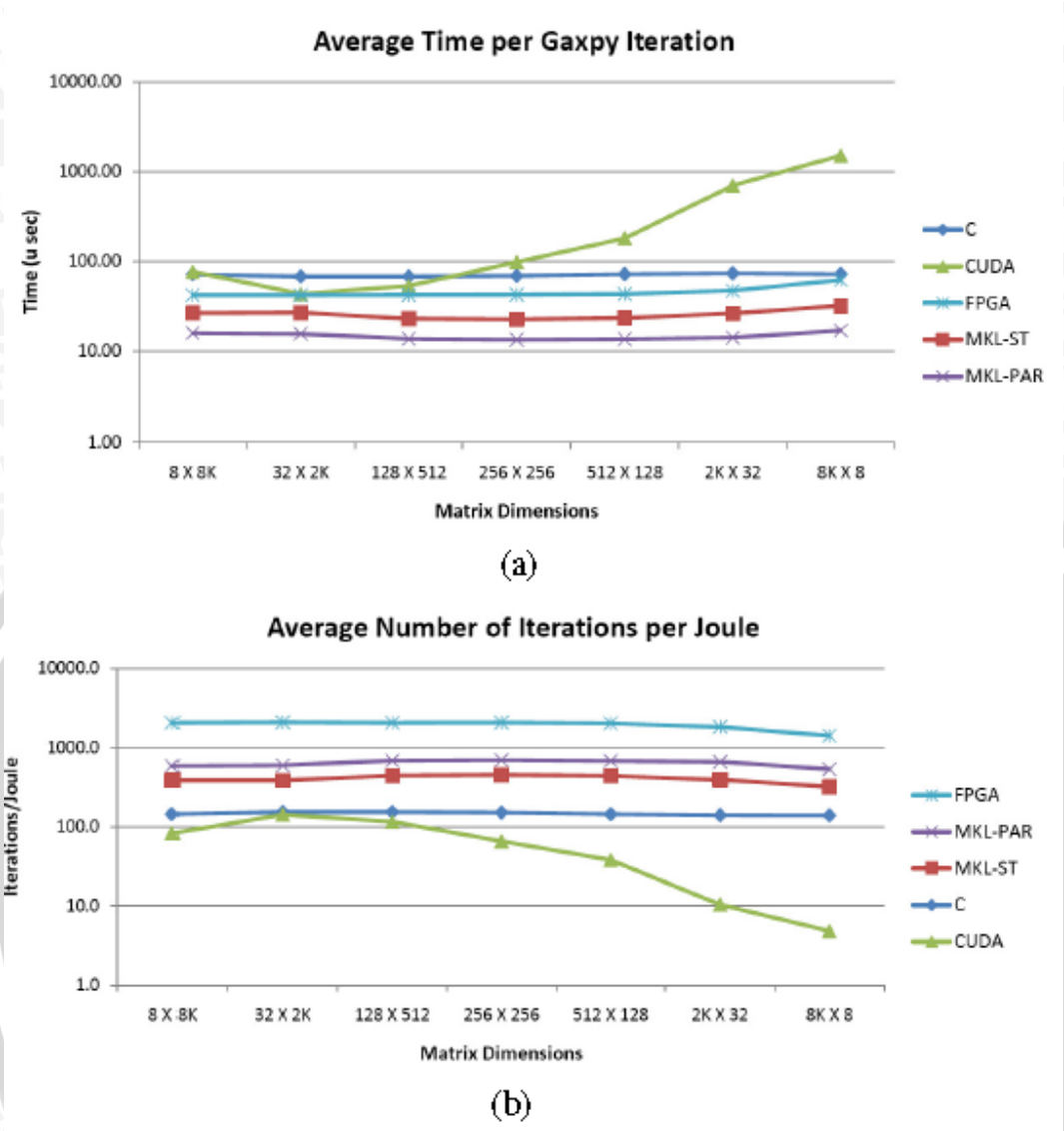
Pada penelitian ini, peneliti melakukan perbandingan performa antara FPGA, CPU dan GPU. Peneliti melakukan perbandingan dalam hal performa dan

energy efficiency. Dari penelitian tersebut mendapatkan kesimpulan bahwa FPGA mempunyai performa yang sama dan efisiensi energi yang lebih tinggi jika dibandingkan dengan CPU dan GPU platform. Selain itu, arsitektur FPGA mampu menyediakan platform yang fleksibel yang dapat menangani bermacam – macam rasio aspek matriks tanpa penurunan kinerja. Gaxpy Performance pada FPGA dapat dilihat pada Gambar 2.2



Gambar 2. 2 Gambar Gaxpy Perfromance on FPGA
Sumber : [Srinidhi-2011]

Gambar 2.2 menggambarkan dampak kinerja perubahan pada waktu pada gaxpy kernel. Implementasi *single-accumulator* terbebankan karena latency dari setiap operasi penyatuan accumulator. Gambar 2.2 juga menggambarkan, *Dual-stage* accumulator adalah desain yang terbaik untuk implentasi Gaxpy FPGA.



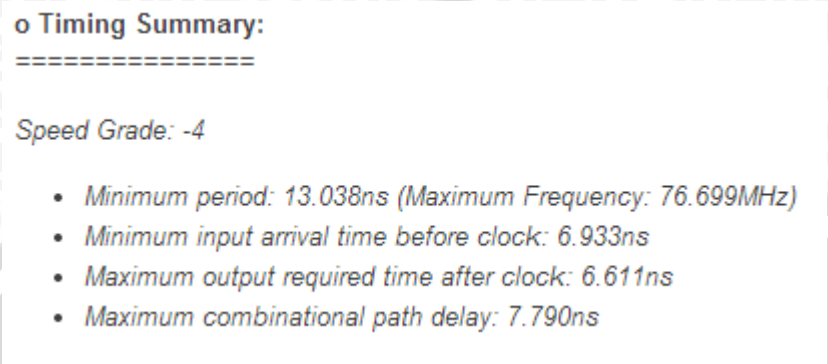
Gambar 2. 3 Perbandingan Gaxpy kernel pada Naïve C, MKL-single thread, MKL-parallel, CUDA BLAS dan FPGA implantation dalam waktu eksekusi (a) dan efisiensi energy (b) Sumber : [Srinidhi-2011]

Gambar 2.3(a) melaporkan waktu eksekusi kernel untuk sebuah macam kombinasi dimensi matriks yang berjalan pada 3 platform. Gambar 2.3(b) menunjukkan energy efisiensi yang dihasilkan pada eksekusi pada setiap platform.

2.1.3 FPGA Implementation of AES Encryption and Decryption oleh Sounak Samanta

Riset mengenai implementasi AES pada FPGA telah dilakukan pada riset yang dilakukan oleh Sounak Samanta, Namun system yang dibuat belum murni

pada FPGA. Pada riset ini Sounak Samanta masih memerlukan bantuan dari PC untuk proses enkripsinya. Pada riset ini proses SubByte dan ShiftRow dilakukan pada PC sedangkan proses MixColumn dan AddRoundKey baru terjadi di FPGA. Hasil yang didapatkan dari riset ini ditunjukkan pada Gambar 2.4



Gambar 2. 4 Hasil riset 2.1.3
Sumber : [Sounak-2009]

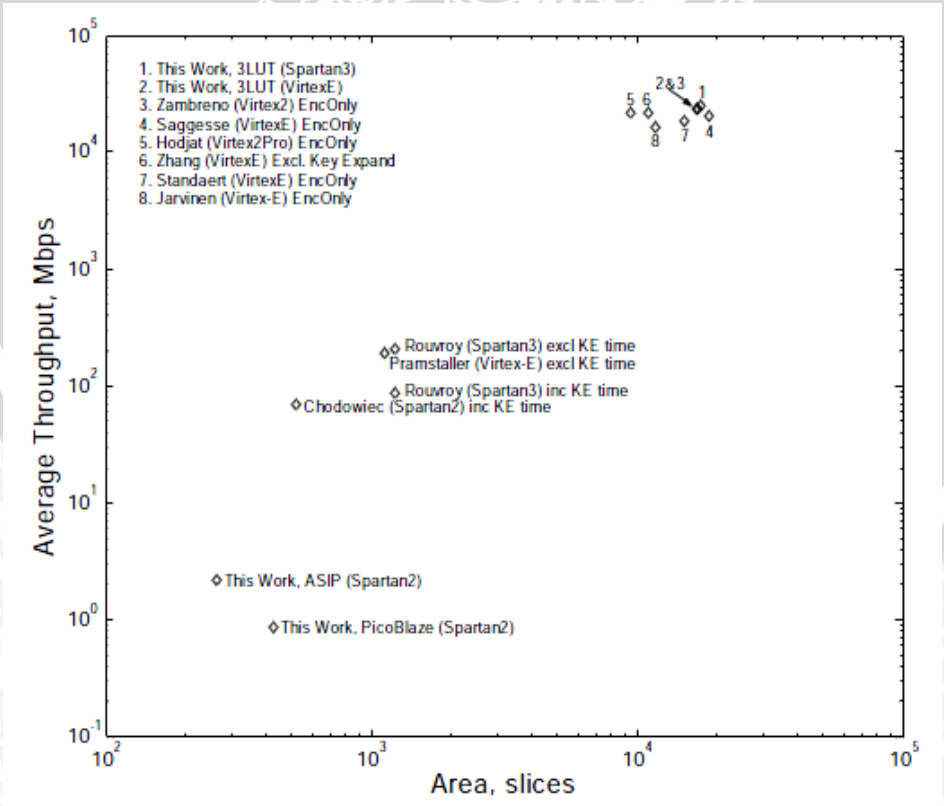
2.1.4 AES on FPGA from the fastest to the smallest oleh Mohammed Benaissa

Pada riset yang dilakukan Mohammed Benaissa lebih menitik beratkan perbandingan pada FPGA yang dipakai dan pada riset ini Mohammed Benaissa hanya memasukkan proses KeyExpansion , MixColumn , dan SubByte saja. Pada riset ini validitas dari hasil enkripsi dikesampingkan hanya menitik beratkan pada FPGA yang effisien dan design pada AES. Perbandingan FPGA yang dipakai terlihat pada Gambar 2.5

	This design	Picoblaze based	Chodowiec & Gaj [4]	Rouvroy et al [5]
FPGA	Spartan-II XC2S15-6	Spartan-II XC2S15-6	Spartan-II XC2S30-6	Spartan-III XC3S50-4
Clock Frequency (MHz)	67	90	60	71
Datapath Bits	8	8	32	32
Slices	124	119	222	163
No. of Block RAMs used	2	2	3	3
Block RAM Size (kbits)	4	4	4	18
Bits of block RAM used	4480	10666	9600	34176
Est. equiv. slices for memory	140	333	300	1068
Total Equiv. Slices (area)	264	452	522	1231
Max Throughput (Mbps)	-	-	166	208
Ave. Throughput (Mbps)	2.2	0.71	69	87
Throughput/slice (kbps/slice)	8.3	1.9	132	70
Summary	Smallest	Software	Best speed/area	Fastest

Gambar 2. 5 Perbandingan design dengan FPGA harga murah
Sumber : [Benaissa – 2009]

Hasil dari riset ini ditunjukkan pada Gambar 2.6.



Gambar 2. 6 Hasil Riset Perbandingan Design FPGA yang berbeda
Sumber : [Benaissa – 2009]

2.1.5 FPGA Implementation of Efficiect Hardware for the Advanced Encryption Standard oleh Amandeep Kaur, Puneet Bhardwaj, Naveen Kumar

Riset yang dilakukan oleh Amandeep Kaur adalah riset tentang implementasi AES pada FPGA, namun hanya sebatas dilakukan pada simulator dan belum dilakukan pengujian tentang validitas serta belum dilakukan pengujian waktu enkripsi yang dilakukan pada AES tersebut. Data hasil riset yang dilakukan ditunjukan pada Gambar 2.7.

S.No.	Signal	Value
1.	keylength	128
2.	decryption	False
3.1	data_in(0)	10101100100101100111000100100101
3.2	data_in(1)	10101001110011101111000011001110
3.3	data_in(2)	00001100000110101111000011010101
3.4	data_in(3)	10101010110001100111100101000111
4	data_stable	1
5.	keyword	10101111000000010011010000100000
6.	keywordaddr	101
7.	w_ena_keyword	1
8.	key_stable	1
9.	decrypt_mode	0
10.	keyexp_done	1
11.1	result(0)	10101111000000010011010000100000
11.2	result(1)	10101111000010010011010000100000
11.3	result(2)	10101111100000010011010000100000
11.4	result(3)	10101111000000010011010000100000
12.	finished	1
13.	round_type_enc	10
14.	finished_enc	1
15.	ena_encrypt	1
16.	key_ready	1
17.	ready	1

Gambar 2. 7 Data Hasil Riset Implementasi AES pada FPGA
Sumber : [Kaur- 2013]

2.2 Dasar Teori

Dasar teori membahas teori yang diperlukan untuk menyusun penelitian yang diusulkan.

2.2.1 Kriptografi

Kriptografi secara umum adalah ilmu dan seni untuk menjaga kerahasiaan berita [**Bruce Schneir** –*Applied Cryptography*]. Selain pengertian tersebut terdapat pula pengertian ilmu yang mempelajari teknik – teknik matematika yang berhubungan dengan aspek keamanan informasi seperti kerahasiaan data, keabsahan data, integritas data, serta autentikasi data. Tidak semua aspek keamanan informasi ditangani oleh kriptografi.

Ada empat tujuan mendasar dari ilmu kriptografi ini yang juga merupakan aspek keamanan informasi yaitu:

1. Kerahasiaan adalah layanan yang digunakan untuk menjaga isi dari informasi dari siapapun kecuali yang memiliki otoritas atas kunci rahasia untuk membuka/mengupas informasi yang telah disandi.
2. Integritas data adalah berhubungan dengan penjagaan dari perubahan data secara tidak sah. Untuk menjaga integritas data, sistem harus memiliki kemampuan untuk mendeteksi manipulasi data oleh pihak – pihak yang tidak berhak, antara lain penyisipan, penghapusan, dan pesubsitusian data lain kedalam data yang sebenarnya.
3. Auntentikasi adalah berhubungan dengan identifikasi/pengenalan, baik secara kesatuan sistem maupun infornasi itu sendiri. Dua pihak yang saling berkomunikasi harus saling memperkenalkan diri. Dua pihak yang saling berkomunikasi harus saling memperkenalkan diri. Informasi yang dikirimkan melalui kanal harus diauntetikasi keaslian isi datanya, waktu pengiriman, dan lain – lain.
4. Non-repudiasi, atau nirpenyangkalan adalah usaha untuk mencegah terjadinya penyangkalan terhadap pengiriman/terciptanya suatu informasi oleh yang mengirimkan/membuat.

2.2.1.1 Terminologi dalam Kriptografi

Ada beberapa istilah – istilah yang penting dalam kriptografi, yaitu:

1. **Pesan** (*Plaintext dan Ciphertext*) : Pesan adalah data atu informasi yang dapat dibaca dan dimengerti maknanya. Pesan asli disebut **plaintext** atau teks jelas. Sedangkan pesan yang sudah disandikan disebut **ciphertext**.
2. **Pengirim** dan **Penerima** : Komunikasi data melibatkan pertukaran pesan antara dua entitas. **Pengirim** adalah entitas yang mengirim pesan kepada entitas lainnya. **Penerima** adalah entitas yang menerima pesan.
3. **Penyadap** (*eavesdropper*) adalah orang yang mencoba menangkap pesan selama ditransmisikan.
4. **Kriptanalisis** dan **Kriptologi** : **Kriptanalisis** adalah ilmu seni untuk memecahkan chiperteks menjadi plainteks tanpa mengetahui kunci yang digunakan. Pelakunya disebut **Kriptanalis. Kriptologi** adalah studi mengenai kriptografi dan kriptanalisis.
5. **Enkripsi** dan **Dekripsi** : Proses menyandikan plainteks menjadi cipherteks disebut **enkripsi**. Sedangkan proses mengembalikan cipherteks menjadi plainteks semula dinamakan **dekripsi**.
6. **Cipher** dan **Kunci** : Algoritma kriptografi disebut juga **cipher** yaitu aturan untuk *enchipering* dan *dechipering,* atau fungsi matematika yang digunakan untuk enkripsi dan dekripsi. **Kunci** adalah parameter yang digunakan untuk transformasi enkripsi dan dekripsi. Kunci biasanya berupa string atau deretan bilangan.

2.2.2 Algoritma Sandi Kunci Simetris

Skema algoritma sandi akan disebut kunci-simetris apabila untuk setiap prose enkripsi maupun dekripsi data secara keseluruhan digunakan kunci yang sama. Skema ini berdasarkan jumlah data per proses dan alur pengolahan data didalamnya dibedakan menjadi dua kelas yaitu blok-cipher dan stream-cipher.

2.2.2.1 Stream-Cipher

Cipher aliran adalah algoritma enkripsi yang mengenkripsi data sebagai aliran karakter-karakter atau bit-bit. Cipher aliran menggunakan dua data aliran data yaitu aliran data masukan dan aliran data kunci semua. Aliran data kunci semua dibangkitkan menggunakan fungsi *pseudo random sequence generator* dimana masukannya adalah kunci induk enkripsi.

Enkripsi pada cipher aliran dilakukan dengan mengkombinasikan satu bit aliran data kunci menggunakan fungsi sederhana tertentu, missal XOR. Proses ini berulang untuk setiap bit aliran data. Keamanan enkripsi lebih sulit ditebak daripada membuat algoritma fungsi yang kompleks. Dekripsi bekerja sebaliknya yaitu dengan mengambil satu bit dan aliran data hasil enkripsi dana satu bit dari data kunci untuk mengembalikan bit aliran data masukan. Ini bekerja karena aliran kunci yang dibangkitkan selalu sama untuk setiap kunci enkripsi dan fungsi enkripsi memiliki operasi *reverse* sendiri yang sederhana.

Contoh algoritma cipher aliran meliputi cipher aliran sinkron dan cipher aliran asinkron. Pada cipher aliran sinkron, kunci dibangkitkan pada waktu yang berbeda selama proses enkripsi. Contoh algoritma cipher aliran sinkron meliputi aliran DES dalam mode OFB. Salah satunya adalah RC4. Cipher aliran asinkron membangkitkan aliran kunci sebagai fungsi dari kunci dan sekumpulan bit cipherteks sebelumnya. Contoh algoritma cipher aliran asinkron adalah aliran DES dalam mode CFB. Salah satunya adalah R5 yang biasa digunakan untuk enkripsi komunikasi via GSM. Contoh lain algoritma enkripsi cipher aliran adalah one-tipe-pad.

2.2.2.2 Blok-Cipher

Cipher Blok adalah algoritma enkripsi dimana data yang dienkripsi dibagi-bagi menjadi blok-blok berukuran sama (biasanya 64 bit), dan setiap blok dienkripsi masing-masing. Algoritma enkripsi ini memerlukan penambahan beberapa bit pada teks masukan dengan tujuan agar ukuran blok terakhir sesuai dengan ukuran blok lainnya. Algoritma dekripsi akan menghapus bit-bit tambahan tersebut sebelum mengembalikan teks hasil enkripsi (cipherteks) menjadi teks masukan semula (plainteks).

Operasi pada cipher blok adalah dengan mengambil sebuah blok dari plainteks dan sebuah blok dari kunci untuk menghasilkan cipherteks. Hal ini dapat dianalogikan menggunakan tabel yang berisi semua kemungkinan blok plainteks sebagai baris tabel dan semua kemungkinan blok kunci sebagai kolom tabel. Tabel ini mendefinisikan fungsi untuk enkripsi. Kemudian ada tabel lain yang mendefinisikan fungsi untuk enkripsi. Kemudian ada tabel lain yang mendefinisikan fungsi untuk dekripsi. Apabila ada blok plainteks M dan blok kunci K, kemudian fungsi enkripsi menghasilkan blok cipherteks O. Apabila melihat baris O dan kolom K dari tabel fungsi dekripsi, maka nilai yang diperoleh adalah M.

2.2.3 Advanced Encryption Standard

Advanced Encryption Standard (AES) merupakan algoritma cryptographic yang dapat digunakan untuk mengamankan data. Algoritma AES adalah blok chipertext simetrik yang dapat mengenkripsi (encipher) dan dekripsi (decipher) informasi. Enkripsi merubah data yang tidak dapat lagi dibaca disebut ciphertext, sebaliknya dekripsi adalah merubah ciphertext data menjadi bentuk semula yang kita kenal sebagai plaintext. Algoritma AES menggunakan kunci kriptografi 128, 192, dan 256 bits untuk mengenkrip dan dekrip data pada blok 128 bits.

2.2.3.1 Input dan Output

Input dan output dari algoritma AES adalah berisikan sequence dari 128 bit (urutan bilangan 0 dan 1). Urutan tersebut berbentuk blok dan jumlah bit menjadi ukuran dari panjangnya. Kunci chipper dari algoritma AES adalah urutan dari 128, 192, dan 256 bit, selain itu maka tidak termasuk kedalam standar AES.

Ukuran dari AES akan selalu dimulai dengan 0 dan diakhiri sesuai dengan jumlah bloknya, 128, 192, atau 256. Misal kita menggunakan variabel i sebagai indek yang berada diantara blok $0 \leq i \leq 128, 0 \leq i \leq 192,$ dan $0 \leq i \leq 256$.

2.2.3.2 Byte

Unit dasar dari algoritma AES adalah byte yang berisikan deretan 8 bit, untuk input, output, atau chipper ditulis sebagai a, byte menghasilkan array yang dapat di bentuk dengan notasi an atau a[n], n sendiri merupakan:

Key length= 128 bits, $0 \leq n < 16$; Block length= 128 bits, $0 \leq n < 16$;

Key length= 192 bits, $0 \leq n < 24$; Block length= 128 bits, $0 \leq n < 24$;

Key length= 256 bits, $0 \leq n < 32$; Block length= 128 bits, $0 \leq n < 32$;

Seluruh nilai dalam byte pada algoritma AES didukung oleh polynomial dari {b7, b6, b5, b4, b3, b2, b1, b0} sehingga dapat dibuat persamaan 2.1:

$$b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0 = \sum_{i=0}^{7} b_i x^i \quad (2.1)$$

Contoh, {01100011} sehingga bisa kita dapat $x^6 + x^5 + x + 1$, nilai ini juga dapat didapat dari hasil pemisahan 4 bit dan melihat table hexadesimal, sehingga dapat menghasilkan {63}, perhatikan Gambar 2.8.

Bit Pattern	Character
0000	0
0001	1
0010	2
0011	3

Bit Pattern	Character
0100	4
0101	5
0110	6
0111	7

Bit Pattern	Character
1000	8
1001	9
1010	a
1011	b

Bit Pattern	Character
1100	c
1101	d
1110	e
1111	f

Gambar 2. 8 Representasi hexadesimal
Sumber : [Rifki-2012]

2.2.3.3 Array dari Byte

Array byte dapat dituliskan sebagai :

$a_1\ a_2\ a_3\ \ldots\ a_{15}$

Jika yang diinput berasal dari 128 bit dituliskan sebagai :

$Input_1\ input_2\ input_3\ \ldots\ input_{126}\ input_{127}$

Jika kita gabungkan maka akan berbentuk:

$a_0 = \{input_0, input_1, \ldots, input_7\}$

$a_1 = \{input_8, input_9, \ldots, input_{15}\}$

…

$a_7 = \{input_{120}, input_{121}, \ldots, input_{127}\}$

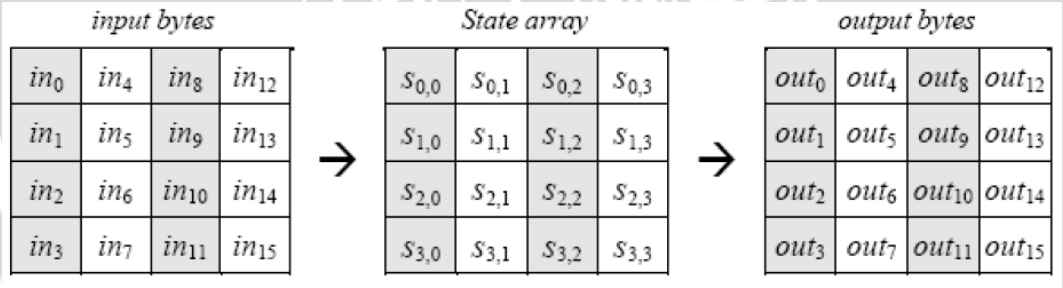
sehingga secara umum :

$a_n = \{input_{8n}, input_{8n+1}, \ldots, input_{8n+7}\}$

2.2.3.4 State

Secara internal, operasi algoritma AES dilakukan pada array 2D dari bytes yang disebut State. State terdiri dari empat baris bytes, masing-masing berisi Nb bytes, dimana Nb adalah panjangnya blok yang dibagi oleh 32. Di dalam Array State yang ditandai oleh lambang, masing-masing byte individu mempunyai dua indeks, dengan jumlah barisnya r didalam rentang $0 \leq r \leq 4$ dan jumlah kolom c di dalam rentang $0 \leq c \leq Nb$. Ini mengijinkan byte State untuk dikenal sebagai sr, c maupun s[r,c]. Karena standard ini, Nb=4, yaitu, $0 \leq c < 4$.

Input – *the array of bytes in0, in1,… in 15* – dimasukan pada State array. Cipher atau kebalikan dari operasi Cipher adalah memasukkan kedalam State array, kemudian langkah akhirnya adalah menyalin hasil pada output – *the array of bytes out0, out1, … out15.*



Gambar 2. 9 State array input dan output.
Sumber : [Rifki-2012]

Pada awal cipher atau kebalikan dari cipher, masukan array disalin kepada State Array menurut persamaan

$s[r,c] = in[r + 4c]$ untuk $0 \leq r < 4$ anda $0 \leq c < Nb$,

dan pada akhir chipper dan kebalikan dari chipper, State disalin kepada output array menurut persamaan:

$out[r + 4c] = s[r , c]$ untuk $0 \leq r < 4$ and $0 \leq c < Nb$.

2.2.4 Spesifikasi Algoritma AES

Algoritma AES mengenkripsikan 128-bit blok plaintext (M), menjadi 123-bit blok chipertext (C), menggunakan kunci chipper (chipper key K) yang panjangnya diantara 128-bit, 192-bit, atau 256-bit. Perbedaan dari panjang kunci menjadikan nama kunci AES-128, AES-192, dan AES-256. Algoritma AES berjalan dalam hitungan byte dan berdasarkan ukuran blok dari input, output dan kuncinya yang direpresentasikan dalam 32-bit words sama dengan 4 byte.

Algoritma AES menggunakan pada Number of Rounds Nr pada saat eksekusinya tergantung pada ukuran dari kunci yang digunakan, seperti terlihat pada Tabel 2.1 :

Tabel 2. 1 Algoritma AES

AES Algorithm	**(Input/Output size) Block Size Nb**	**Key Length N_k**	**Number of Rounds N_r**
AES-128	4 words	4 words	10 rounds
AES-192	4 words	6 words	12 rounds
AES-256	4 words	8 words	14 rounds

Sumber : [Rifki-2012]

Setiap rounds berisi 4 byte-oriented cryptographic transformation :

1. Byte substitution
2. Shifting rows of the state array
3. Data Mixing antara kolom dengan state array
4. Round keyaddition pada state array

2.2.4.1 SubBytes() Transformation

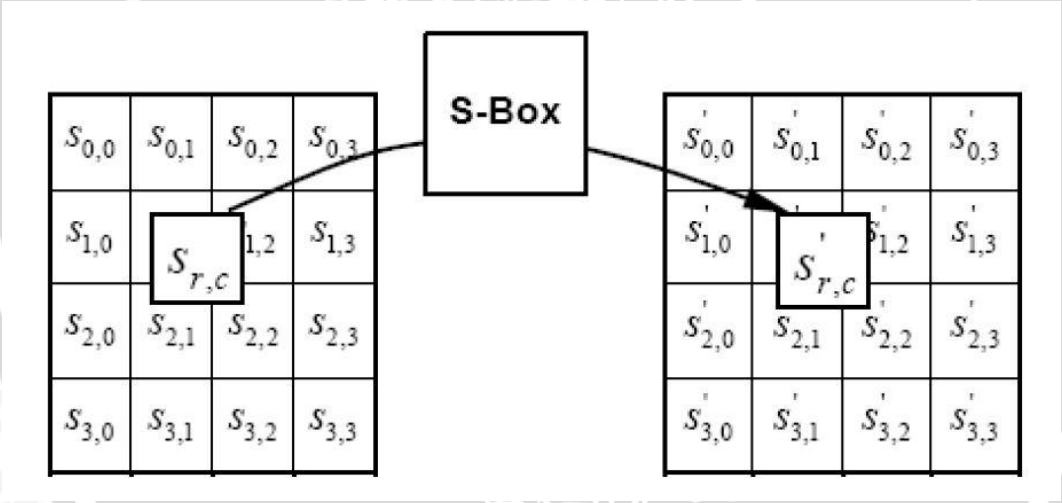
Transformasi pertama pada AES adalah transformasi non-linier byte substitution yang disebut sebagai transformasi SubBytes(). SubBytes() beroperasi dengan bebas pada masing-masing byte State menggunakan table subtitusi (S-box).

Dalam bentuk matriks, elemen S-Box dapat dilihat pada Gambar 2.10 :

$$\begin{bmatrix} b_0' \\ b_1' \\ b_2' \\ b_3' \\ b_4' \\ b_5' \\ b_6' \\ b_7' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

Gambar 2. 10 Elemen S-Box
Sumber : [Rifki-2012]

Gambar 2.11 menggambarkan efek SubBytes() perubahan bentuk pada State.



Gambar 2. 11 Efek SubBytes()
Sumber : [Rifki-2012]

S-box digunakan pada transformasi SubBytes() diperlihatkan dalam bentuk hexadecimal, Sebagai contoh, jika $s_{1,1}$= {53}, kemudian nilai subtitusi akan ditentukan oleh persimpangan antara baris dengan index '5' dan kolom dengan index '3'. Ini akan menghasilkan $s_{1,1}$ = {ed}

		y														
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
	1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
	2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
	3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
	5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
	7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
	8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
	9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
	a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
	b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
	c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
	e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
	f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Gambar 2. 12 SubBytes.
Sumber : [Rifki-2012]

2.2.4.2 ShiftRows() Transformation.

Pada transformasi ShiftRows(), byte pada tiga baris terakhir digeser sebanyak ofsetnya, sedangkan byte pada baris pertama tidak mengalami penggeseran.

$\mathbf{S}_{r,c} = \mathbf{S}_{r,(c+shift(r, Nb)) \bmod Nb}$ Untuk $0 < r < 4$ dan $0 \leq c < Nb$

Pergeseran byte tergantung pada nilai pada barisnya r, (untuk Nb = 4):

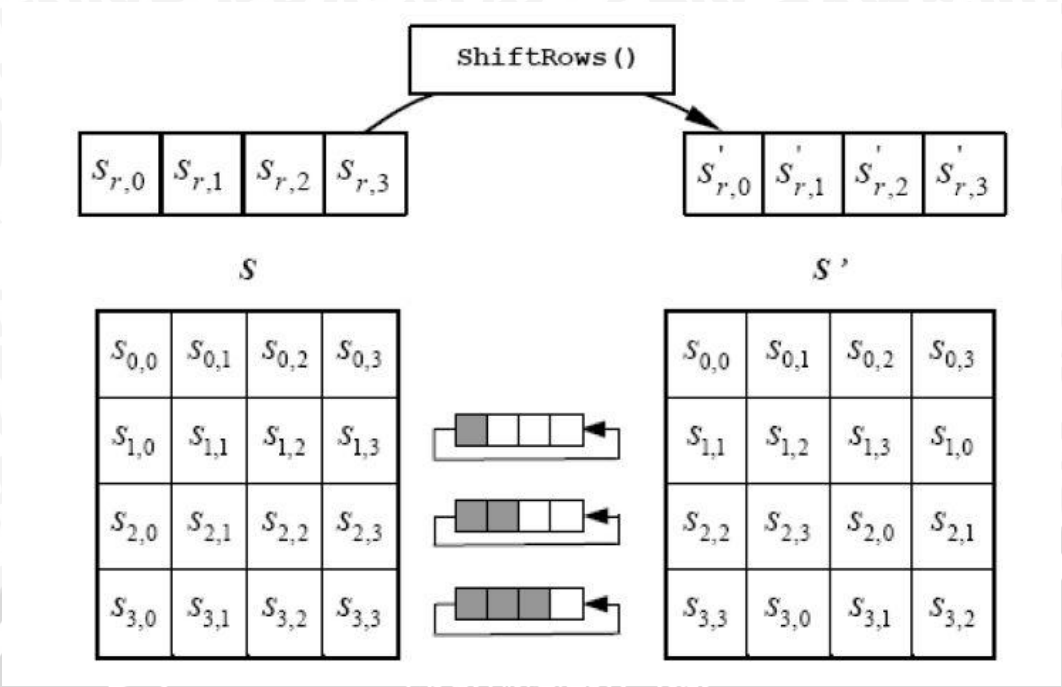
Shift(0,4) = 0 ; bergeser 0 langkah atau tidak bergeser

Shift(1,4) = 1 ; bergeser 1 langkah

Shift(2,4)= 2 ; bergeser 2 langkah

Shift(3,4)= 3 ; bergeser 3 langkah

Pergeseran dilakukan dari depan ke belakang, perhatikan ilustrasi transformasi ShiftRows() dan gambar 2.13 menunjukan bagaimana ShiftRows() bergeser



Gambar 2. 13 ShiftRows.
Sumber : [Rifki-2012]

2.2.4.3 MixColumns() Transformation.

Transformasi MixColumns() digunakan untuk mencampur kolom dari matrik State, Kolom direpresentasikan sebagai polynomial Galois Field GF(2^8). Output dari transformasi MixColumns() s(x), seperti pada persamaan berikut:

$$s`(x) = a(x) \otimes s(x) \bmod (x^4 + 1)$$

Dimana a(x) = $03x^3 + 01x^2 + 02$ kemudian dapat digambarkan sebagai matrik seperti pada gambar 2.14

$$\begin{bmatrix} S'_{0,c} \\ S'_{1,c} \\ S'_{2,c} \\ S'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} S_{0,c} \\ S_{1,c} \\ S_{2,c} \\ S_{3,c} \end{bmatrix}$$

Gambar 2. 14 Gambar matrik mixColumns
Sumber : [Rifki-2012]

Untuk $0 \leq c < Nb$

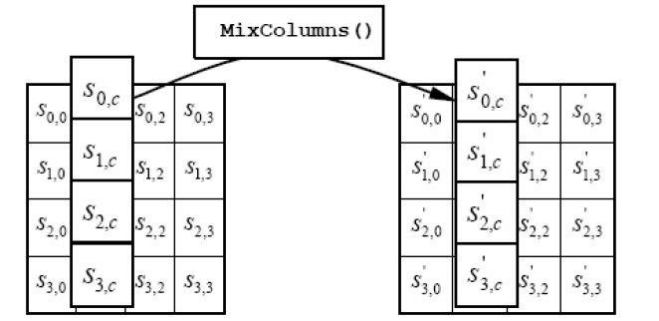
Hasil dari perkalian pada matrik tersebtu dapa dilihat pada persamaan – persamaan seperti pada gambar 2.15 :

$$s'_{0,c} = (\{02\} \bullet s_{0,c}) \oplus (\{03\} \bullet s_{1,c}) \oplus s_{2,c} \oplus s_{3,c}$$

$$s'_{1,c} = s_{0,c} \oplus (\{02\} \bullet s_{1,c}) \oplus (\{03\} \bullet s_{2,c}) \oplus s_{3,c}$$

$$s'_{2,c} = s_{0,c} \oplus s_{1,c} \oplus (\{02\} \bullet s_{2,c}) \oplus (\{03\} \bullet s_{3,c})$$

$$s'_{3,c} = (\{03\} \bullet s_{0,c}) \oplus s_{1,c} \oplus s_{2,c} \oplus (\{02\} \bullet s_{3,c}).$$



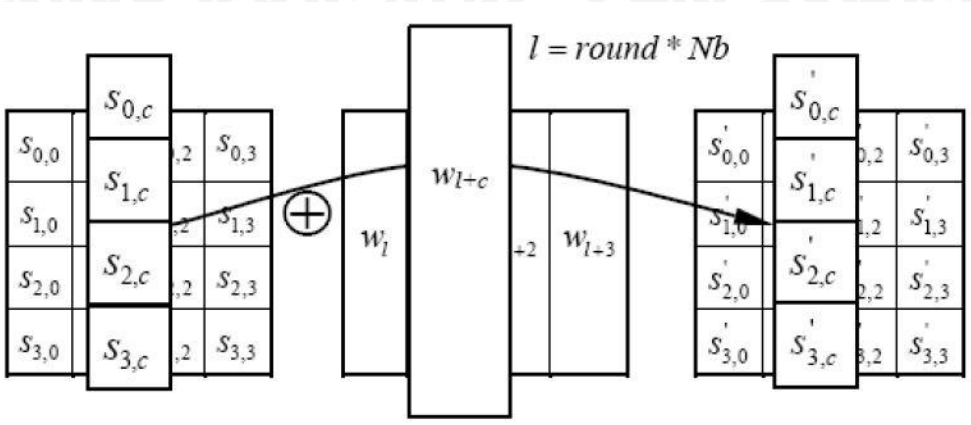
Gambar 2. 15 transformasi MixColumns() pada state Column-to-column
Sumber : [Rifki-2012]

2.2.4.4 AddRoundKey() Transformation.

Kunci dari bit dihasilkan dari kunci cipher yang asli dengan menggunakan transformasi key expansion yang dijumlahkan bit pada array dari state dengan operasi XOR yang sederhana. Nilai awal dari round key adalah w0, untuk round = 0, ditambahkan pertama kalo para cryptographic round. Kemudian setiap round untuk $1 \leq$ round $\leq$ Nr, kemudian 32-bit yang berbeda pada round-key wi ditambahkan.

$$[s'_{0,c}, s'_{1,c}, s'_{2,c}, s'_{3,c}] = [s_{0,c}, s_{1,c}, s_{2,c}, s_{3,c}] \oplus [w_{round*Nb+c}] \quad (2.2)$$

Untuk $0 \leq c < Nb$.



Gambar 2. 16 ilustrasi transformasi AddRoundKey()
Sumber : [Rifki-2012]

2.2.5 Field Programable Gate Array (FPGA)

FPGA adalah perangkat elektronik semikonduktor dalam sebuah IC yang dapat dikonfigurasi oleh programmer untuk keperluan simulasi sistem. Papan FPGA sering juga disebut pemodelan atau prototyping pemrograman perangkat keras dimana pengguna dapat menggunakan sesuai dengan kebutuhan.

FPGA terdiri dari sejumlah gerbang yang dapat di program untuk tujuan tertentu. Komponen gerbang yang terdapat dalam sebuah FPGA meliputi gerbang-gerbang logika dasar seperti OR, AND, NOT, XOR dan sejumlah rangkaian kombinasional seperti decoder, penjumlahan, pengali dll. Terdapat juga bagian untuk menyimpan data seperti register, flip-flop atau sel memori lainnya. Antar komponen dalam FPGA dihubungkan oleh sistem interkoneksi yang dirancang sedemikian rupa sehingga memudahkan programmer mengkonfigurasi ulang untuk keperluan rancangan tertentu, itulah kenapa disebut *programmable.*

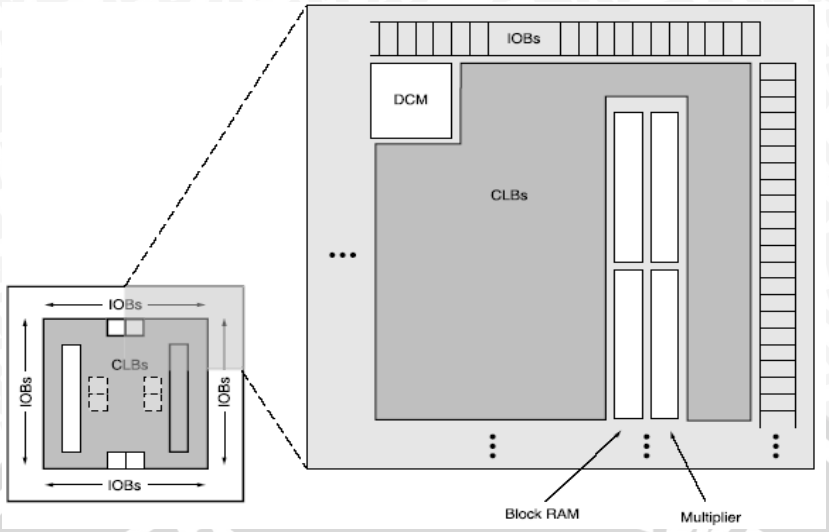
Secara umum FPGA akan lebih lambat jika dibandingkan dengan chip yang lain seperti chip *Application-specific Integrated Circuit* (ASCI). Hal ini karena FPGA menggunakan daya yang besar dan bentuk desai yang komplek. Beberapa kelebihan FPGA adalah harganya yang murah, bisa deprogram sesuai dengan kebutuhan dan bisa deprogram ulang untuk mengkoreksi adanya kesalahan pada program. Beberapa karakteristik FPGA adalah sebagai berikut.

A. Mudah dikonfigurasi ulang artinya FPGA dapat deprogram sesuai dengan kebutuhan. Program yang tersimpan dalam FPGA bersifat sementara dan dapat diubah setiap saat.
B. Kecepatan tinggi, FPGA terdiri dari sekumpulan perangkant keras dengan kecepatan clock yang tinggi, karena itu aplikasi yang tertanam di dalamnya memiliki kecepatan yang tinggi
C. Perlindungan hak cipta dan pengguna ulang rancangan. Ketika sebuah rancangan diterapkan pada FPGA, aplikasinya tidak dapat dikembalikan lagi ke dalam bentuk program, jadi terlindung dari orang-orang yang tidak bertanggung jawab. Program yang sudah dirancang dapat digunakan ulang kapanpun waktu diperlukan.

FPGA Spartan 3e adalah salah satu dari keluarga FPGA yang di produksi oleh Xilinx yang dapat dilihat di Gambar 2.17.

Gambar 2. 17 FPGA Spartan 3e
Sumber: [Maman-2012]

Bahasa pemrograman yang digunakan dalam FPGA ini adalah VHDL (*Very High Hardware Description Language*) dengan bantuan *software development kit* dari Xilinx. Secara umum arsitektur dalam IC FPGA terdiri dari tiga bagian yaitu *Input Output Block* (IOB), *Configurable Logic Block* (CLB), dan Interkoneksi (kanal-kanal routing). Arsitektur dalam IC FPGA Spartan 3e dapat dilihat di Gambar 2.18.



Gambar 2. 18 Arsitektur FPGA Spartan 3e,
Sumber: [Maman-2012]

1. *Input Output Block* (IOB) mengontrol aliran data antara I/O pin dan logika internal. Setipa IOB juga mendukung bagian standar sinyal.
2. *Configurable Logic Block* (CLB) memiliki Look-Up Tables (LUTs) yang fleksibel dimana dapat mengimplementasikan logika dan ada tempat penyimpanan yang digunakan sebagai flip-flop atau latches. CLB melakukan berbagai fungsi logika dan juga penyimpanan data.
3. Blok RAM menyediakan penyimpanan data dalam bentuk dual port blok 19 Kbit.
4. Block Multiplier menerima dua angka biner 18-bit sebagai input dan menghitungnya.
5. Digital block Manager menyediakan kalibrasi, delay, multipliying, dividing dan fase pergeseran sinyal clock.

Semua bagian dari arsitektur ini teroganisir dengan baik seperti yang di tunjukan pada gambar (Arsitektur FPGA Spartan 3e). Cincin IOBs mengelilingi array CLBs. Setiap RAM terdiri dari beberapa blok 18Kbit RAM. Setiap blok RAM dikaitkan dengan multiplier. DCMs diposisikan di tengan dengan dua di atas dan dua di bawah.

2.2.6 Bahasa Pemrograman Hardware

Dalam bahasa pemrograman perangkat keras (*hardware*) ada beberapa bahasa yang umum digunakan untuk pemrograman IC yaitu Verilog dan VHDL.

Diantara kedua kode tersebut memiliki perbedaan yang cukup signifikan. Namun secara filosofi konsep, perbedaan dasar dari VHDL dengan Verilog adalah mengenai konteks dari kedua bahasa itu sendiri. Verilog berasal dari tradisi "bottom-up" yang telah sering digunakan dalam industry IC dalam hal rancangan dasar IC. Sedangkan kode VHDL dikembangkan lebih kepada persepektif "top-down". Tentu saja, banyak perbedaan umum dan luas dalam konteks saat ini. Namun, secara jelas dan nyata, perbedaannya dapat terlihat pada syntax dasar dan metode dari kedua kode tersebut. Satu hal penting tentang keunggulan VHDL adalah kemampuannya untuk menggunakan gabungan lebel dari model yang memiliki arsitektur yang berbeda. Hal tersebut memang bukanlah keunikan atau ciri khas VHDL. Namun, pada kenyataannya kode Verilog juga memiliki konsep sama walaupun hanya terdapat dalam sebuah "module". Meskipun demikian, keunggulan itu secara eksplisit didefinisikan dalam VHDL dan secara praktis digunakan bersama oleh rancangan multi level dalam VHDL.

VHSIC *Hardware Description Language* (VHDL) merupakan sebuah bahasa pemrograman dari VHSIC (*Very High Speed Integrated Circuit*) yang dikembangkan oleh IEEE (*Institute Electrical and Elektronik Engineering*) pada tahun 1987, yang digunakan untuk merancang dan memodelkan sebuah rangkaian digital. VHDL telah mengalami modifikasi dan revisi dengan versi terbaru berlabel IEEE std 1076-1993.

Desain digital yang akan digunakan dalam VHDL digambarkan dengan menggunakan eksternal view dengan satu atau beberapa view. Eksternal view adalah interface dari rancangan sedangkan internal view menyatakan fungsi atau struktur dari rancangan yang dibuat, biasanya satu rancangan memiliki satu atau lebih view.

VHDL memungkinkan untuk menggambarkan sistem digital di structural, pada umumnya structural tersebut dibagi menjadi dua yaitu flow dan algoritmik. Data flow merupakan representasi yang menggambarkan bagaimana data bergerak melalui sistem, biasanya ini dilakukan dalam aliran data antara register. Aliran data penggunaan bersamaan membuat pernyataan yang dijalankan secara parallel

data segera setelah tiba di masukan. Sedangkan pernyataan sekuensial dijalankan dalam urutan yang mereka tetapkan.

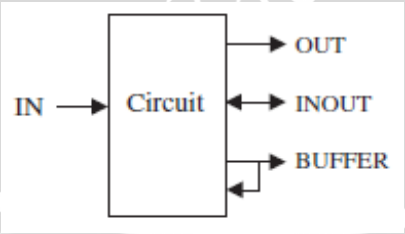
Secara umum pendeskripsian VHDL terdapat dua bagian, yaitu arsitektur dan entity. Bagian lainnya adalah library, sinyal, variabel, konstan, array, fungsi dan prosedur. Entity memberikan arti tentang bagaiman sebuah bagian rancangan dideskripsikan di VHDL dalam hubungannya dengan model VHDL lain dan juga memberikan nama untuk model tersebut. Didalam entity juga diperbolehkan untuk mendefinisikan beberapa parameter yang mengambil model menggunakan hierarki. Contoh sintaks entity seperti yang ditunjukan pada Gambar 2.19.

```
ENTITY entity_name IS
   PORT (
      port_name : signal_mode signal_type;
      port_name : signal_mode signal_type;
      ...);
END entity_name;
```

Gambar 2. 19 Penulisan Sintax Entity
Sumber : [Maman-2012]

Sinyal *mode* bisa IN, OUT, INOUT, atau BUFFER. Seperti digambarkan dalam gambar 2.19, IN dan OUT adalah pin searah, sedangkan INOUT adalah pin dua arah. BUFFER digunakan ketika sinyal output harus digunakan internal.

Sinyal *type* bisa BIT, STD_LOGIC, INTEGER, dll. *Nama* pada entity pada dasarnya bisa menggunakan nama apapun. Architecture digunakan untuk mendeskripsikan rangkaian yang akan dibuat. Penulisan architecture ditunjukkan pada Gambar 2.20.



Gambar 2. 20 Gambar Signal Mode
Sumber : [Maman-2012]

```
ARCHITECTURE architecture_name OF entity_name IS
    [declarations]
BEGIN
    (code)
END architecture_name;
```

Gambar 2. 21Gambar Penulisan Architecture
Sumber : [Maman-2012]

Seperti yang ditunjukan Gambar 2.21, arsitektur memiliki dua bagian: bagian *deklaratif* (opsional), di mana sinyal dan konstanta (antara lain) dinyatakan, dan bagian *kode* (dari BEGIN ke bawah). Seperti dalam kasus entity, nama arsitektur pada dasarnya dapat nama apapun (kecuali kata-kata VHDL yang telah dipesan), termasuk nama yang sama dengan entity.

Pendeskripsian VHDL dapat dilakukan dengan menggunakan metode *behavioral dan structural.*

A. Behavioral

   Arsitektur dapat didesain sesuai dengan prinsip kerja alat. Kunci dalam behavior adalah proses. Proses adalah kumpulan dari sequential statement yang dipasang parallel dengan statemen yang sama dan proses yang lain.

B. Structural

   Arsitektur structural menghubungkan modul-modul yang ada pada sistem FPGA. Arsitektur structural mendefinisikan struktur interkoneksi yang tepat dari sinyal dan entity.

Bagian lain yang juga penting dalam arsitektur adalah proses dan konfigurasi. Proses adalah sub dari sebuah arsitektur. Dalam suatu arsitektur bisa terdapat satu atau lebih proses. Sedangkan konfigurasi digunakan untuk mendeklarisakan kumpulan desain entity tertentu sebagai komponen.

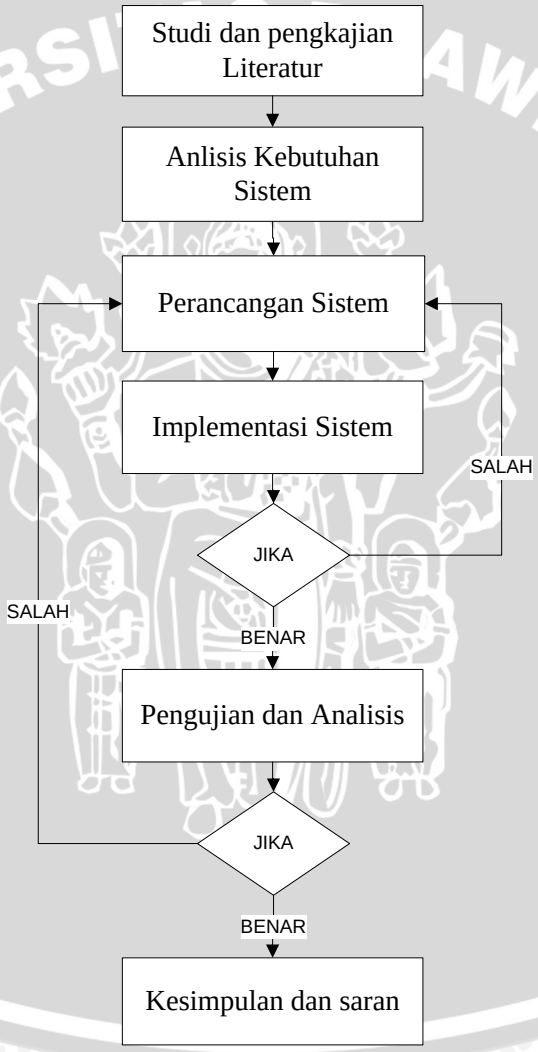
A. Sinyal = dapat dianalogikan sebagai kabel yang menjadi penghubung antar bagian dalam sistem yang dirancang. Sinyal dideklarasikan dalam arsitektur.

B. Variable = memiliki nilai yang langsung berubah, tanpa harus menunggu selesainya suatu proses. Variable dideklarasikan didalam proses.
C. Fungsi = dapat menerima suatu data masukan dengan tipe data tertentu dan menghasilkan keluaran yang telah didefinisikan dalam spesifikasinya.
D. Prosedur = hampir sama dengan fungsi, namun prosedur tidak dapat mengembalikan nilai balikan. Prosedur dalam VHDL harus menyertakan argument dengan spesifikasi dengan tipe out atau inout.

BAB III
METODOLOGI PENELITIAN DAN PERANCANGAN

3.1 Metodologi Peneitian

Bab ini menjelaskan langkah-langkah yang akan ditempuh dalam penyusunan skripsi, meliputi studi dan pengkajian literature, analisis kebutuhan sistem, perancangan sistem, implementasi sistem, pengujian dan analisis, kesimpulan dan saran.



Gambar 3. 1 Flowchart Metode Penelitian

3.1.1 Studi Literatur

Dalam perancangan dan implementasi penelitian ini, perlu diadakan studi literatur. Literatur digunakan sebagai teori penguat dan landasan dasar dalam penelitian. Teori pendukung tersebut di dapat dari buku, jurnal, paper dan internet. Literatur yang digunakan meliputi :

1. Kriptografi
    a. Jenis- jenis Algoritma Kriptografi
    b. Stream cipher
    c. Block cipher
2. Algoritma *Advanced Encryption Standard*
    a. SuBytes() Transformation
    b. ShiftRows() Transformation
    c. MixColumns() Transformation
    d. AddRoundKey() Transformation
3. Embedded System
    a. FPGA
    b. FPGA Spartan 3e
    c. Pin Out *FPGA Spartan 3e*
    d. 16x2 LCD display
    e. Keyboard input

3.1.2 Analisis Kebutuhan Sistem

Analisis kebutuhan bertujuan untuk mendapatkan semua kebutuhan yang diperlukan oleh sistem yang akan dibangun dan di uji. Analisis kebutuhan dilakukan dengan mengidentifikasi kebutuhan sistem. Analisis kebutuhan tersebut meliputi, kebutuhan perangkat lunak dan kebutuhan perangkat keras.

Sesuai dengan tujuan dari penulisan ini yang mengarah kearah *embedded system* yang dimana suatu sistem berbasis mikroprosessor yang dibangun untuk mengendalikan satu atau beberapa fungsi dan tidak dirancang untuk dapat deprogram oleh pengguna akhir seperti computer pirbadi maka dibutuhkan

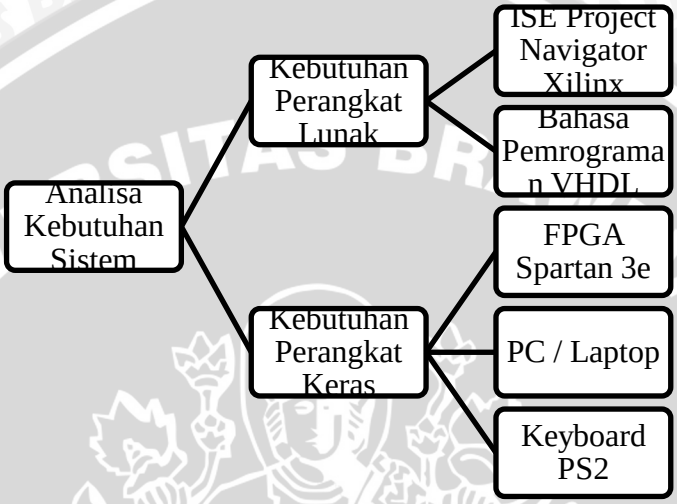
kebutuhan perangkat yang sesuai. Kebutuhan perangkat pada penelitian ini adalah pada penelitian ini dibutuhkan *embedded system* yang dapat bekerja secara cepat dan efisien dalam mengolah algoritma AES yang cukup rumit. Dari penelitian – penelitian sebelumnya muncul nama FPGA yang bekerja lebih effisien dan lebih berperforma dibandingkan CPU dan GPU, selain itu FPGA jika dibandingkan dengan *embedded system* yang lainnya misalnya Rasphberry atau arduino, FPGA berjalan pada layer physical sehingga proses eksekusinya lebih cepat sedangkan Rasphberry atau Arduino berjalan pada layer aplikasi. Dari pertimbangan tersebut maka dipilih FPGA untuk menerapkan penelitian ini.

Pada penelitian ini dibutuhkan juga LCD untuk menampilkan hasil dari enkripsi data teks, juga dibutuhkan port yang mengubungkan Keyboard yang nantinya berfungsi sebagai inputan. Pada FPGA Spartan 3e sudah dilengkapi dengan LCD 16 x 2 dan port PS2 untuk menghubungkan keyboard. Dari pertimbangan tersebut makan dipilih FPGA Spartan 3e untuk menjadi board implementasi pada penelitian ini.

Pada penelitian ini untuk merancang system dibutuhkan sebuah PC ataupun laptop. Kebutuhan perangkat lunak digunakan untuk merancang program enkripsi data text dengan algoritma AES. Untuk mendesain dan merancang system dibutuhkan compiler yang cocok dengan hardware yang digunakan yaitu Spartan 3e. Xilinx Spartan 3e memiliki software bawaan yang sudah pasti cocok dengan FPGA tersebut dan nantinya juga bisa digunakan untuk mengimportkan program kedalam FPGA. Maka dari itu dipilih ISE Project Navigator sebagai compiler dan sebagai desain program system. Pada kebutuhan perangkat lunak yang lainnya adalah bahasa pemrograman. Pada penelitian ini dibutuhkan bahasa pemrograman yang dapat merancang sistem digital yang komplek.

Pada penelitian ini juga dibutukan bahasa pemrograman yang dapat menggunakan gabungan level dari model yang memiliki arsitektur yang berbeda. System yang dibuat nantinya memiliki multiple behavior dalam sebuah interface. Selain itu model tersebut memungkinkan terjadi pertukaran dan implementasi multiple secara terus menerus. Dilihat dari kebutuhan bahasa pemrograman tersebut maka VHDL lah yang cocok untuk kebutuhan tersebut, karena

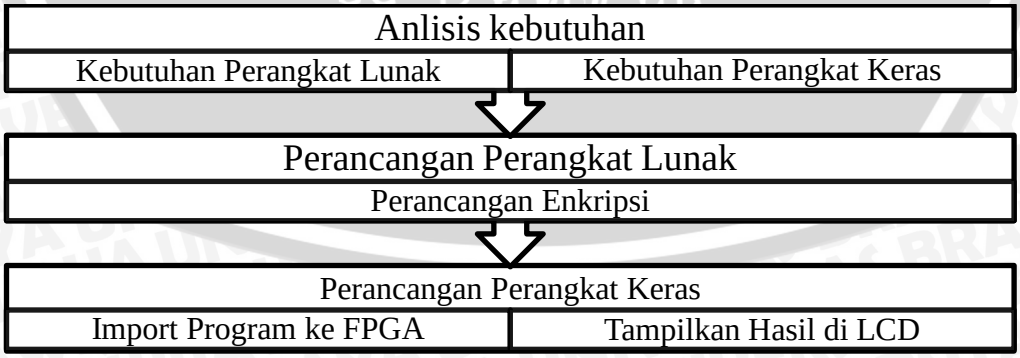
karakteristik VHDL adalah mempunyai kemampuan yang lebih untuk merancang sistem digital yang komplek. Satu hal penting tentang keunggulan VHDL dibandingkan verilog adalah kemampuannya untuk menggunakan gabungan level dari model yang memiliki arsitektur yang berbeda. Dari kebutuhan – kebutuhan tersebut maka dirancanglah diagram kebutuhan sistem seperti yang ditunjukkan pada Gambar 3.2



Gambar 3. 2 Diagram kebutuhan sistem

3.1.3 Perancangan Sistem

Perancangan sistem dilakukan setelah semua kebutuhan sistem didapatkan melalui tahap analisis kebutuhan yang meliputi kebutuhan perangkat lunak dan kebutuhan perangkat keras. Setelah kebutuhan sistem terpenuhi selanjutnya adalah perancangan perangkat lunak dan perancangan perangkat keras. Seperti yang di tunjukan dalam Gambar 3.3.



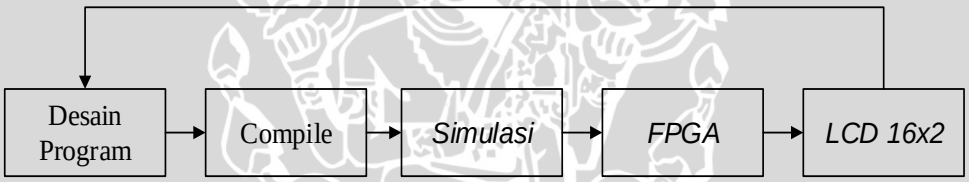
Gambar 3. 3 Alur perancangan Sistem secara umum

Pada perancangan enkrispi dilakukan dengan algoritma *Advanced Encryption Standard* (AES). Algoritma AES adalah blok chipertext simetrik yang dapat

mengenkripsi (encipher) informasi. AES beroperasi berdasarkan prinsip desain yang diketahui sebagai jaringan substitusi – permutasi. Dengan desain ini, AES dapat diimplementasikan dengan efektif baik pada hardware maupun software. Algoritma AES dispesifikasikan sebagai jumlah perulangan dari transformasi loop yang mengkonversi plaintext menjadi ciphertext. Masing-masing loop terdiri atas beberapa proses, termasuk satu langkah yang bergantung pada kunci enkripsinya. Pada perancangan perangkat keras terdapat proses import program ke FPGA dan ditampilkan dalam LCD 16 x 2 pada proses tersebut digunakan ISE Project Navigator sebagai perantara atau penghubung dalam setiap proses tersebut.

3.1.4 Implementasi

Implementasi merupakan penjelasan mulai dari perancangan program sampai hasil akhir yang di tampilkan dalam LCD Monitor. Seperti yang di gambarkan dalam Gambar 3.4.



Gambar 3. 4 Gambaran secara umum implementasi enkripsi data text dengan Algoritma AES.

Desain program, *compile* sampai *simulate* di jalankan di perangkat lunak pendukung bawaan dari Xilinx. Jika program sukses, *import* program dalam papan FPGA yang kemudian hasilnya akan ditampilkan dalam LCD 4 bit. Proses ini mungkin tidak berjalan satu kali, Jika dirasa program belum sesuai kebutuhan, program akan di desain kembali dalam perangkat lunak.

3.1.5 Pengujian dan Analisis

Pengujian dilakukan dengan 3 tahap :

1. Pengujian fungsional
2. Pengujian validitas hasil Enkripsi system
3. Pengujian waktu eksekusi

Pengujian yang pertama adalah pengujian fungsional. Pengujian ini dilakukan untuk mengetahui masukan dan hasil keluaran itu selaras. Pengujian ini dilakukan dengan cara membandingkan inputan dan output yang dikeluarkan.

Pengujian kedua adalah pengujian validitas hasil Enkripsi system. Pengujian dilakukan dengan cara membandingkan hasil cipherteks dari system yang dibuat dengan hasil cipherteks yang dihasilkan oleh kalkulator AES dengan kunci yang sama. Jika memang sistem memberikan hasil yang valid maka seharusnya hasil yang dikeluarkan maka akan sama dengan hasil yang dikeluarkan oleh kalkulator AES. Dari pengujian ini akan diambil 10 Data yang akan di uji.

Pengujian ketiga adalah pengujian waktu. Pengujian ini dilakukan untuk mengetahui waktu eksekusi yang dibutuhkan sistem untuk mengenkripsi data. Pengujian ini akan dilakukan dengan cara memasukkan beberapa sampel data yang berbeda panjang karakternya. Hasil dari setiap panjang karakter akan dibandingkan dan di analisa.

3.1.6 Kesimpulan

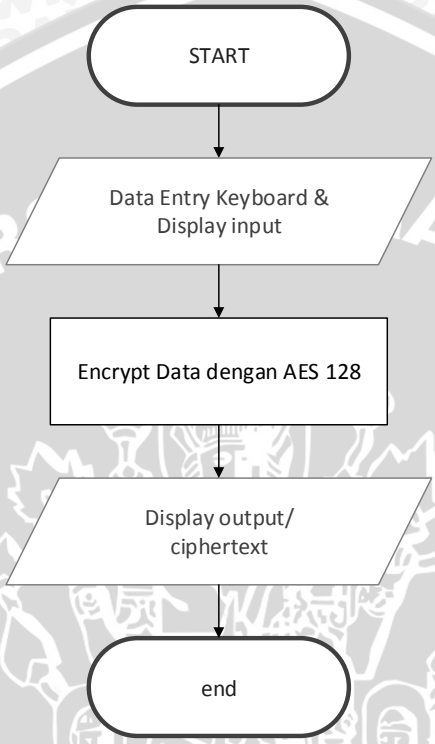
Pengambilan kesimpulan dilakukan setelah semua tahapan perancangan, implementasi, pengujian dan analisis sistem telah selesai dilakukan. Kesimpulan diambil dari hasil pengujian dan analisis terhadap sistem yang dibuat. Kesimpulan di ambil untuk mengetahui validitas algoritma AES.

Tahap akhir yang dilakukan adalah saran untuk memperbaiki kesalahan-kesalahan yang terjadi dalam pembuatan enkripsi teks dengan algoritma AES pada FPGA dan menyempurnakan penulisan serta untuk memberikan pertimbangan atas pengembangan aplikasi selanjutnya.

3.2 Perancangan

3.2.1 Perancangan Enkripsi

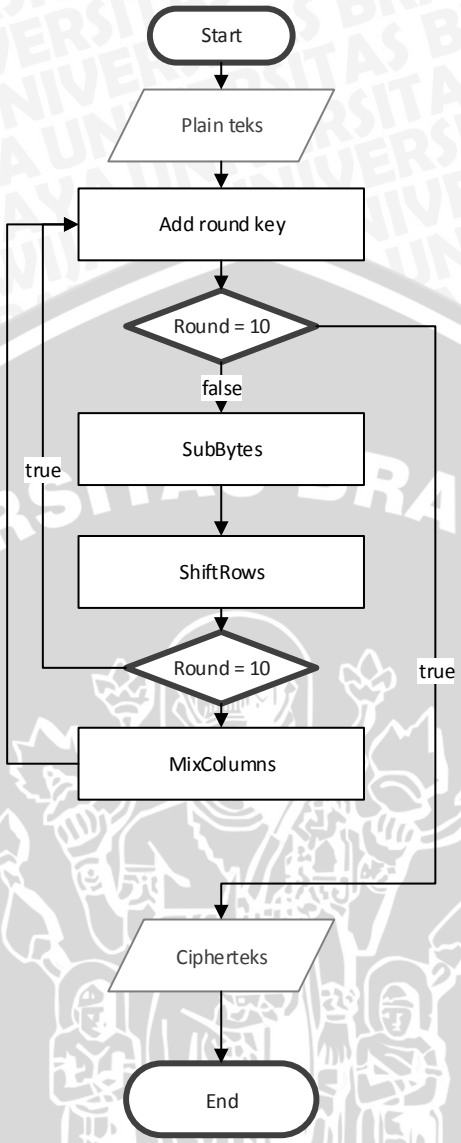
Perancangan enkripsi merupakan bagian perancangan perangkat lunak. Enkripsi data teks dilakukan dengan algoritma AES. Diagram alir dari proses umum system dijelaskan pada Gambar 3.5



Gambar 3. 5 Diagram alir proses umum system

A. Proses Encrypt Data dengan AES 128

Proses enkripsi dilakukan dengan algoritma AES 128 bit. Flowchart proses enkripsi akan dijelaskan pada Gambar 3.6



Gambar 3. 6 Flowchart Enkripsi AES 128 bit

Pada Gambar 3.6 menjelaskan proses enkripsi dengan algoritma AES 128 bit. Algoritma enkripsi AES 128 dijelaskan pada tabel 3.1

Tabel 3. 1 Algoritma Enkripsi AES

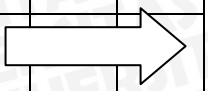
Nama algoritma : Algorima Enkripsi AES 128 bit

Input : P, K {Teks Asli 16 bytes , Kunci AES}
Output : CT { Teks Sandi 16 bytes}
Proses :
(Nr, w) ← **Ekspansi Kunci**(K) { Nr : Jumlah Round , w : larik bytes kunci round}
CT = P
AddRoundKey(CT, w[0…3])
For i = 1 → Nr **do**
SubBytes(CT)
ShiftRows(CT)
if i /= Nr **Then**
MixColumns(CT)
End if
AddRoundKey(CT, w[(i*4)…(i*4)+3])
End for

B. SubBytes

Pada proses SubBytes akan dilakukan substitusi nilai hex pada nilai awal dengan tabel SBOX. SBox yang digunakan adalah SBox standard AES. Tabel SBox yang dipakai akan ditunjukan pada Gambar 2.6. Proses SubBytes dapat dilihat dari proses data pada Gambar 3.7

19	a0	9a	e9
3d	f4	c6	F8
E3	E2	8d	48
be	2b	2a	08



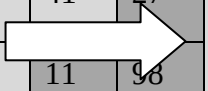
D4	E0	B8	1e
27	bf	B4	41
11	98		
ae	F1	E5	30

Gambar 3. 7 Data Proses SubByte

C. ShiftRows

Pada proses ShiftRows akan dilakukan dengan menjalankan operasi *circular shift left* sebanyak i pada baris ke-i pada *state.* Proses berjalannya ShiftRows akan ditunjukkan pada Gambar 2.7. Hasil dari proses Shift Rows bisa dilihat pada Gambar 3.8.

D4	E0	B8	1e
27	bf	B4	41
11	98	5d	52
ae	F1	E5	30



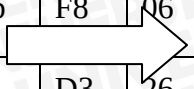
D4	E0	B8	1e
Bf	B4	41	27
5d	52	11	98
30	ae	F1	E5

Gambar 3. 8 Data Proses ShiftRows

D. MixColumns

Pada Proses MixColumn akan dilakukan mencampur nilai kolom – kolom pada state pada satu elemen pada state keluaran. Untuk melakukan pencampuran itu, transformasi MixColumn menggunakan operasi perkalian matriks dengan operasi perkalian dan penjumlahan menggunakan operator pada GF(2^8) dengan *irreducible polynomial.* Proses MixColumn akan ditunjukan pada Gambar 2.9. Proses dari Mix column dapat dilihat pada Gambar 3.9

D4	E0	B8	1e
Bf	B4	41	27
11	98	5d	52
ae	F1	E5	30



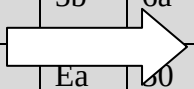
04	E0	48	28
66	Cb	F8	06
81	19	D3	26
35	9a	7a	4c

Gambar 3. 9 Data Proses mix Column

E. AddRoundKey

Transformasi AddRoundKey mencampur sebuat state masukan dengan kunci ronde dengan operasi ekslusif OR. Setiap elemen pada state masukan yang merupakan sebuat byte dikenakan operasi ekslusif OR dengan byte pada posisi yang sama di kunci ronde. Proses AddRoundKey akan ditunjukan pada Gambar 2.10. Data dari hasil Add round key bisa dilihat pada Gambar 3.10 dan kunci round nya pada Gambar 3.11

04	E0	48	28
66	Cb	F8	06
81	19	D3	26
35	9a	7a	4c



A4	68	6b	02
9c	9f	5b	6a
7f	35	Ea	30
F2	2b	43	49

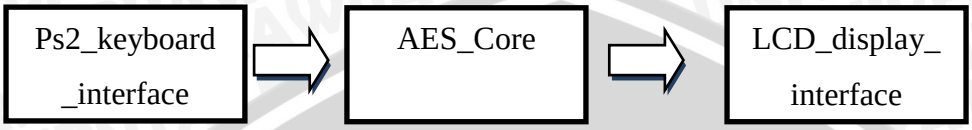
Gambar 3. 10 data hasil AddRoundKey

A0	88	23	2a
Fa	54	A3	6c
Fe	2c	39	76
17	B1	39	05

Gambar 3. 11 Kunci Round

3.2.2 Perancangan Input dan Output Program

Perancangan Input dan Output Program akan ditunjukan pada Gambar 3.12.



Gambar 3. 12 Perancangan Input Output Program

Pada Gambar 3.12 menunjukkan bahwa inputan program akan dimasukkan pada keyboard dahulu, lalu diolah pada AES_Core. Setelah diolah di AES_Core dengan menerjemahkan inputan keyboard yang dalam bentuk hex akan diterjemahkan ke ASCII sehingga dapat diterjemahkan dalam LCD_display_interface dan pada akhirnya akan ditampilkan sesuai dengan ASCII yang telah dikirim dari AES_Core.

3.2.3 Perancangan Perangkat Keras

Berdasarkan pada pohon perancangan pada Gambar 3.2. Perancangan perangkat keras di lakukan setelah analisis kebutuhan dan perancangan perangkat lunak selesai dilakukan. Perancangan perangkat keras ditunjukan pada Gambar 3.12.

Gambar 3. 13 Perancangan Perangkat Keras

Perancangan perangkat keras dilakukan setelah perancangan perangkat lunak selesai dilakukan. Seperti yang sudah disebutkan didalam analisi kebutuhan

perangkat keras, pada perancangan perangkat keras dibutuhkan laptop/PC, FPGA Spartan 3e, Keyboard PS2 dan USB.

Setelah desain program selesai dan program sukses langkah selanjutnya adalah mengimportkan program kedalam FPGA Spartan 3e, pada proses ini dibutuhkan kabel USB untuk menghubungkan antara laptop dengan FPGA Spartan 3e. Setelah program sukses diimportkan didalam papan FPGA Spartan 3e, langkah terakhir adalah menampilkan hasil program yang sudah diimportkan di papan FPGA Spartan 3e kedalam Lcd 16x2 yang sudah ada di papan FPGA Spartan 3e.

BAB IV
IMPLEMENTASI

4.1 Lingkungan Implementasi

Lingkungan implementasi dari enkripsi teks dengan algoritma AES pada FPGA meliputi lingkungan perangkat lunak (Software) dan lingkungan perangkat keras (Hardware).

4.1.1 Lingkungan Perangkat Lunak

Lingkungan perangkat lunak yang digunakan dalam mengimplementasikan enkripsi teks dengan algoritma AES adalah sebagai berikut:

1. Sistem operasi windows 7 ultimate 64-bit sebagai proses implementasi.
2. ISE Project Navigator 13.4 merupakan *software development* bawaan dari Xilinx Spartan 3e yang digunakan untuk pemrograman.

4.1.2 Lingkungan Perangkat Keras

Lingkungan perangkat keras yang digunakan dalam mengimplementasikan enkripsi teks dengan algoritma AES pada FPGA adalah sebagai berikut:

1. FPGA Xilinx Spartan 3e
2. Kabel USB
3. Laptop prosesor intel® core™ i3 CPU M330
4. RAM 4GB
5. Keyboard PS2

4.2 Batasan Implementasi

Beberapa batasan implementasi dalam implementasi teks dengan algoritma AES pada FPGA sebagai berikut :

1. Enkripsi data text di implementasikan di papan FPGA Spartan 3e.
2. Implementasi dititik beratkan pada proses Enkripsi.
3. Panjang karakter 16 karakter untuk plain text dan 32 karakter yang ditampilkan pada cipher text dalam bentuk hex.

4. Algoritma enkripsi yang digunakan adalah algoritma AES 128bit.

4.3 Implementasi Perangkat Lunak

Implementasi enkripsi teks dengan algoritma AES pada FPGA berdasarkan perancangan sistem pada BAB III. Pada implementasi perangkat lunak yaitu implementasi enkripsi. Bahasa yang digunakan adalah bahasa VHDL.

4.3.1 Implementasi Enkripsi

Pada proses implementasi enkripsi terdapat beberapa proses yaitu SubBytes, ShiftRows, MixColumns, dan AddRoundKey. SourceCode pada riset ini akan ditampilkan pada lampiran tersendiri.

A. SubBytes

AES menggunakan substitusi non linear pada ukuran byte yang disebut dengan SubBytes. Transformasi SubBytes menggunakan sebuah tabel substitusi. Proses SubBytes akan dijelaskan pada pseudocode 4.1

```
Start
    Status <= "100"
    MAIN_counter <= MAIN_counter+1
    case MAIN_counter is
        Case 0 :   MAIN_counter <= 1;
                    address<= "100" & INT_REGS(0,0)
         Case 1 :     INT_REGS(0,0)<=rombus
         Case 2 :     address<="100" & INT_REGS(1,0)
         Case 3 :      INT_REGS(1,0)<=rombus
         Case 4 :      address<="100"& INT_REGS(2,0)
         Case 5 :      INT_REGS(2,0)<=rombus
         Case 6 :      address<="100"& INT_REGS(3,0)
         Case 7 :      INT_REGS(3,0)<=rombus
    End case
Stop
```

PseudoCode 4.2 : Proses SubBytes

B. ShiftRows

Selain menggunakan substitusi untuk mengganti nilai pada elemen *state*, AES menggunakan permutasi pada *state*. ShiftRows dilakukan dengan menjalan operasi *circular shift left* sebanyak i pada baris ke-i pada *state*. Proses ShiftRows pada program akan dijelaskan pada PseudoCode 4.3.

```
Start
    when "0110" =>  --shift rows
    status<="100"
    MAIN_counter<=MAIN_counter+1
     case MAIN_counter is
         Case 0 :
          tempbyte1<=INT_REGS(1,0);
          tempbyte2<=INT_REGS(2,0);
          tempbyte3<=INT_REGS(3,0);
          tempbyte4<=INT_REGS(2,1);
          tempbyte5<=INT_REGS(1,2);
          tempbyte6<=INT_REGS(3,2);
        Case 1:
          INT_REGS(1,0)<=INT_REGS(1,1);
          INT_REGS(2,0)<=INT_REGS(2,2);
          INT_REGS(3,0)<=INT_REGS(3,3);
          INT_REGS(2,1)<=INT_REGS(2,3);
          INT_REGS(1,2)<=INT_REGS(1,3);
          INT_REGS(3,2)<=INT_REGS(3,1);
        Case 2:
          INT_REGS(1,3)<=tempbyte1;
          INT_REGS(2,2)<=tempbyte2;
          INT_REGS(3,1)<=tempbyte3;
          INT_REGS(2,3)<=tempbyte4;
          INT_REGS(1,1)<=tempbyte5;
```

```
            INT_REGS(3,3)<=tempbyte6;
          Case 3 :
           MAIN_counter<=0;
           status<="010";
          Case others :
           status<="101";
    end case;
Stop
```

PseudoCode 4.3 : Proses ShiftRows

C. MixColumns

Tujuan transformasi MixColumn adalah mencampur nilai kolom pada *state* pada satu elemen pada *state* keluaran. Proses MixColumn pada system akan ditunjukan pada PeudoCode 4.4.

```
Start
        when "0111" =>  -- MixColumns
        t       status<="100";
              MAIN_counter<=MAIN_counter+1;
              case MAIN_counter is
              Case 0 :
                          inv0(8 downto 0):=INT_REGS(0,0) & '0' xor
                          INT_REGS(1,0) & '0' xor
                          '0' & INT_REGS(1,0) xor
                          '0' & INT_REGS(2,0) xor
                          '0' & INT_REGS(3,0);
                          inv1(8 downto 0):=INT_REGS(1,0) & '0' xor
                          INT_REGS(2,0) & '0' xor
                          '0' & INT_REGS(2,0) xor
                          '0' & INT_REGS(0,0) xor
                          '0' & INT_REGS(3,0);
                          inv2(8 downto 0):=INT_REGS(2,0) & '0' xor
                          INT_REGS(3,0) & '0' xor
```

```
                    '0' & INT_REGS(3,0) xor
                    '0' & INT_REGS(0,0) xor
                    '0' & INT_REGS(1,0);
                    inv3(8 downto 0):=INT_REGS(3,0) & '0' xor
                    INT_REGS(0,0) & '0' xor
                    '0' & INT_REGS(0,0) xor
                    '0' & INT_REGS(1,0) xor
                    '0' & INT_REGS(2,0);
if(inv0(8)='1') then
        inv0(7 downto 0):=inv0(7 downto 0)  xor "00011011";
end if;
if(inv1(8)='1') then
         inv1(7 downto 0):=inv1(7 downto 0)  xor "00011011";
end if;
   if(inv2(8)='1') then
        inv2(7 downto 0):=inv2(7 downto 0)  xor "00011011";
end if;
   if(inv3(8)='1') then
         inv3(7 downto 0):=inv3(7 downto 0)  xor "00011011";
end if;
                    INT_REGS(0,0)<=inv0(7 downto 0);
                    INT_REGS(1,0)<=inv1(7 downto 0);
                    INT_REGS(2,0)<=inv2(7 downto 0);
                    INT_REGS(3,0)<=inv3(7 downto 0);
          MAIN_counter<=1;
Stop
```

PseudoCode 4.4 : Proses MixColumn

D. AddRoundKey

Transformasi keempat yang digunakan pada penyandian AES adalah transformasi AddRoundKey. Transformasi AddRoundKey mencampur sebuat

state masukan dengan kunci ronde dengan operasi eksklusif OR. Proses AddRoundKey pada program akan di jelaskan pada PseudoCode 4.5

```
Start
        when "0100" => -- add round key
       status<="100";
        MAIN_counter<=MAIN_counter+1;
       case MAIN_counter is
              Case 0 :
                  if(inverse='1') then
                        if(WORD_EXTREGPTR=0) then
                              WORD_EXTREGPTR<=80;
                   elsif(WORD_EXTREGPTR=15) then
                         WORD_EXTREGPTR<=0;
                  end if;
                  end if;
             Case 1 :
                    address <= "010" & IntTouVec(WORD_EXTREGPTR,8);
             Case 2 :
                    tempbyte1<=INT_REGS(0,0);
                    tempbyte2<=INT_REGS(1,0);
             Case 3 :
                    INT_REGS(0,0)<=rambus(15 downto 8) xor tempbyte1;
                    INT_REGS(1,0)<=rambus(7 downto 0)  xor tempbyte2;
                   WORD_EXTREGPTR<=WORD_EXTREGPTR+1;
            Case 4 :
                   address<="010" & IntTouVec(WORD_EXTREGPTR,8);
      Case 5 :
                    tempbyte1<=INT_REGS(2,0);
                   tempbyte2<=INT_REGS(3,0);
      Case 6 :
                    INT_REGS(2,0)<=rambus(15 downto 8) xor tempbyte1;
```

```
                INT_REGS(3,0)<=rambus(7 downto 0) xor tempbyte2;
                WORD_EXTREGPTR<=WORD_EXTREGPTR+1;
        Case 7 :
                address <= "010" & IntTouVec(WORD_EXTREGPTR,8);
        Case 8 :
                tempbyte1<=INT_REGS(0,1);
                tempbyte2<=INT_REGS(1,1);
        Case 9 :
                INT_REGS(0,1)<=rambus(15 downto 8) xor tempbyte1;
                INT_REGS(1,1)<=rambus(7 downto 0) xor tempbyte2;
                WORD_EXTREGPTR<=WORD_EXTREGPTR+1;
        Case 10 :
                address<="010" & IntTouVec(WORD_EXTREGPTR,8);
        Case 11 :
                 tempbyte1<=INT_REGS(2,1);
                 tempbyte2<=INT_REGS(3,1);
        Case 12 :
                INT_REGS(2,1)<=rambus(15 downto 8) xor tempbyte1;
                INT_REGS(3,1)<=rambus(7 downto 0) xor tempbyte2;
                WORD_EXTREGPTR<=WORD_EXTREGPTR+1;
        Case 13 :
                address <= "010" & IntTouVec(WORD_EXTREGPTR,8);
        Case 14 :
                 tempbyte1<=INT_REGS(0,2);
                 tempbyte2<=INT_REGS(1,2);
        Case 15 :
                 INT_REGS(0,2)<=rambus(15 downto 8) xor tempbyte1;
                 INT_REGS(1,2)<=rambus(7 downto 0) xor tempbyte2;
                 WORD_EXTREGPTR<=WORD_EXTREGPTR+1;
        Case 16 :
                 address<="010" & IntTouVec(WORD_EXTREGPTR,8);
```

```
Case 17 :
            tempbyte1<=INT_REGS(2,2);
            tempbyte2<=INT_REGS(3,2);
Case 18 :
            INT_REGS(2,2)<=rambus(15 downto 8) xor tempbyte1;
            INT_REGS(3,2)<=rambus(7 downto 0) xor tempbyte2;
            WORD_EXTREGPTR<=WORD_EXTREGPTR+1;
Case 19 :
            address <= "010" & IntTouVec(WORD_EXTREGPTR,8);
Case 20 :
           tempbyte1<=INT_REGS(0,3);
           tempbyte2<=INT_REGS(1,3);
Case 21 :
           INT_REGS(0,3)<=rambus(15 downto 8) xor tempbyte1;
           INT_REGS(1,3)<=rambus(7 downto 0) xor tempbyte2;
           WORD_EXTREGPTR<=WORD_EXTREGPTR+1;
Case 22 :
           address<="010" & IntTouVec(WORD_EXTREGPTR,8);
Case 23 :
           tempbyte1<=INT_REGS(2,3);
           tempbyte2<=INT_REGS(3,3);
Case 24 :
           INT_REGS(2,3)<=rambus(15 downto 8) xor tempbyte1;
           INT_REGS(3,3)<=rambus(7 downto 0) xor tempbyte2;
Case 25:
  MAIN_counter<=0;
  if(inverse='0') then
           if(WORD_EXTREGPTR<87) then
             WORD_EXTREGPTR<=WORD_EXTREGPTR+1;
           else
             WORD_EXTREGPTR<=0;
```

```
                    end if;
            else
                    if(WORD_EXTREGPTR>16) then
                      WORD_EXTREGPTR<=WORD_EXTREGPTR-15;
                    elsif(WORD_EXTREGPTR=15) then
                    else
                      WORD_EXTREGPTR<=0;
                    end if;
             end if;
            status<="010";
      Case others :
                    rambus<=(others =>'Z');
                    status<="101";
      end case;
Stop
```

PseudoCode 4.5 : Proses AddRoundKey

4.3.2 Implementasi Input Output

Pada proses implementasi Input Output terdapat beberapa SubProgram Keyboard_PS2, AES_Core, dan LCD_Display_Interface. SourceCode pada riset ini akan ditampilkan pada lampiran tersendiri.

A. Keyboard_PS2

Pada SubProgram Keyboard_PS2 berfungsi untuk membaca inputan dari perangkat keyboard sehingga program dapat menerima data dari keyboard dan menerjemahkannya. Implementasi Keyboard_PS2 dapat dilihat pada PseudoCode 4.6

```
Start
Read input keboard
IF (“tombol ditekan” = “huruf dipilih”)
Then ASCII_AES_Core = “ASCII huruf dipilih”
END IF
```

```
END
```

PseudoCode 4.6 : SubProgram Keyboard_PS2

B. AES_CORE

Pada SubProgram AES_Core adalah otak dari sistem ini. AES_Core bukan hanya mengatur pada proses enkripsi saja namun juga mengatur inputan dan keluaran pada program. Pada sisi input dan output akan di jelaskan pada PseudoCode 4.7.

```
Start
ASCII = ASCII_keyboard
IF (counter = 0)
INT_REGS(0,0) = ASCII
Core2LCD = INT_REGS(0,0)
Counter += 1
End if
End
```

PseudoCode 4.7 : SubProgram AES_Core proses Input Output

C. LCD_Display_Interface

Pada SubProgram LCD_Display Interface berfungsi untuk mengatur keluaran pada LCD hardware. Proses LCD_Display_interface akan dijelaskan pada PseudoCode 4.8

```
Start
Byte = Core2LCD
Display = Byte
End
```

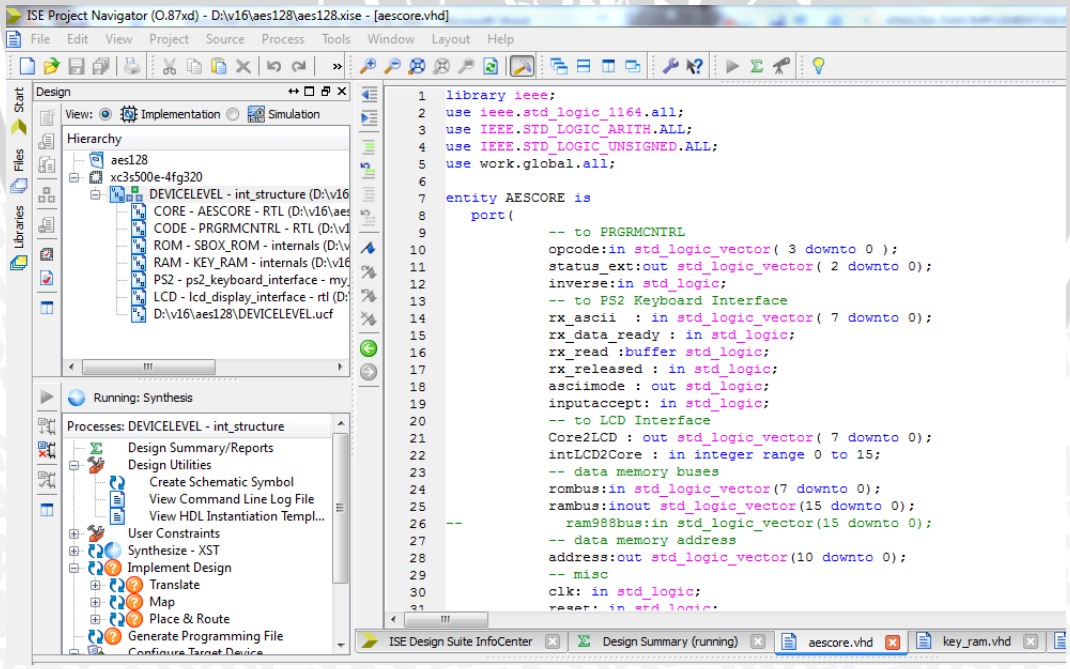
PseudoCode 4.8 : SubProgram LCD_Display_Interface

4.4 Implementasi Perangkat Keras

Implementasi enkripsi teks dengan algoritma AES pada FPGA berdasarkan perancangan sistem pada BAB III. Implementasi perangkat keras yaitu implementasi import ke FPGA dan setelah itu langsung di tampilkan di Lcd 16x2 yang sudah terdapat di Spartan 3e keluaran Xilinx.

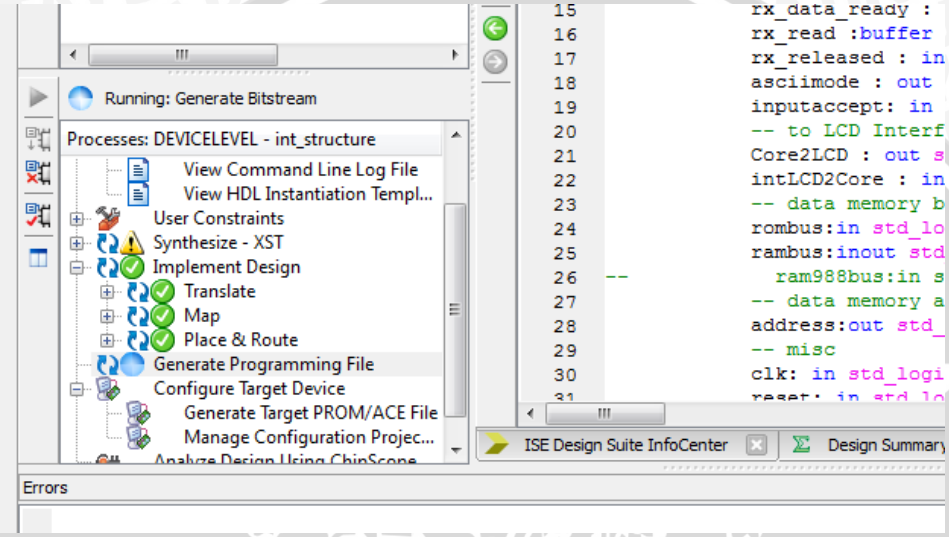
4.4.1 Implementasi Import Program Kedalam FPGA

Tahap awal yang dilakukan sebelum import kedalam FPGA adalah memastikan bahwa program yang telah di desain didalam ISE Project Navigator selesai, berhasil dijalankan dan tidak error dalam program. Proses compile program seperti yang ditunjukan pada Gambar 4.1.



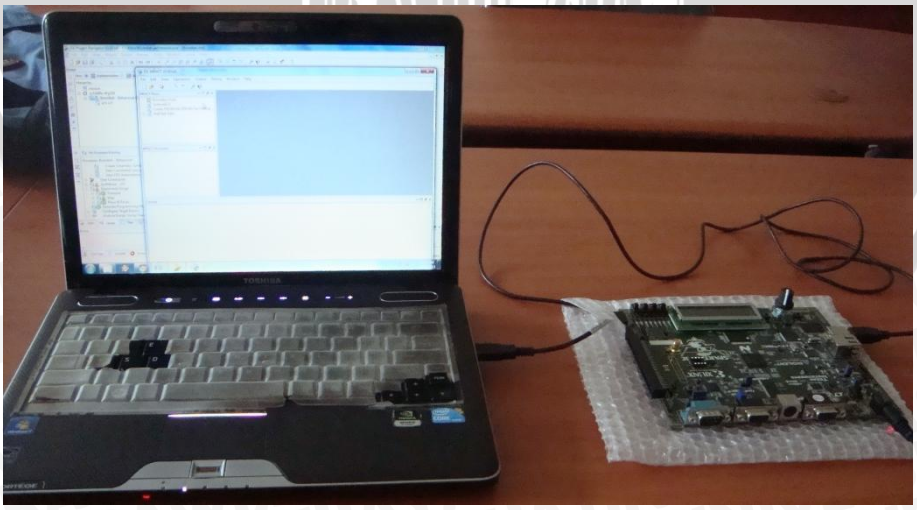
Gambar 4. 1 Proses Compile Program

Program pada synthesize, translate, map, place and route dan generate program harus berhasil dan berwarna hijau atau berwarna kuning. Warna kuning menunjukan dalam program masih beberapa warning namun warning dalam program tidak menggagalkan program dalam artian program bisa diimportkan dalam FPGA. Tanda selesainya program dan siap di importkan dalam FPGA ditunjukan dalam Gambar 4.2.



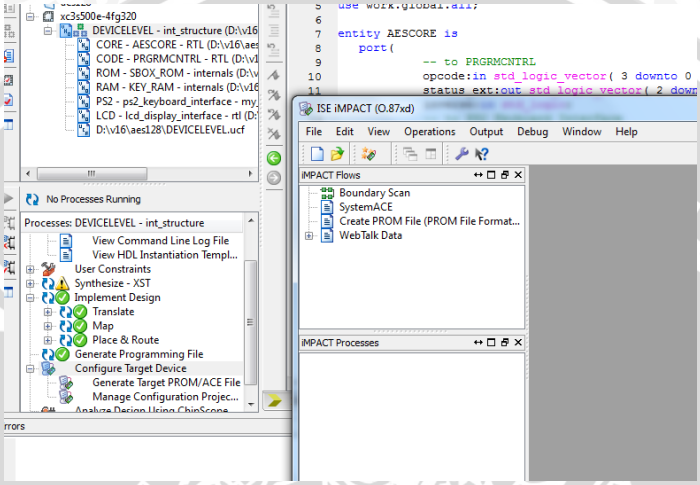
Gambar 4. 2 Program sukses

Setelah itu maka dilakukan proses import program didalam FPGA. Dalam hal ini dibutuhkan kabel USB untuk menghubungkan antara laptop dengan FPGA Spartan 3e. Desainnya ditunjukan dalam Gambar 4.3.



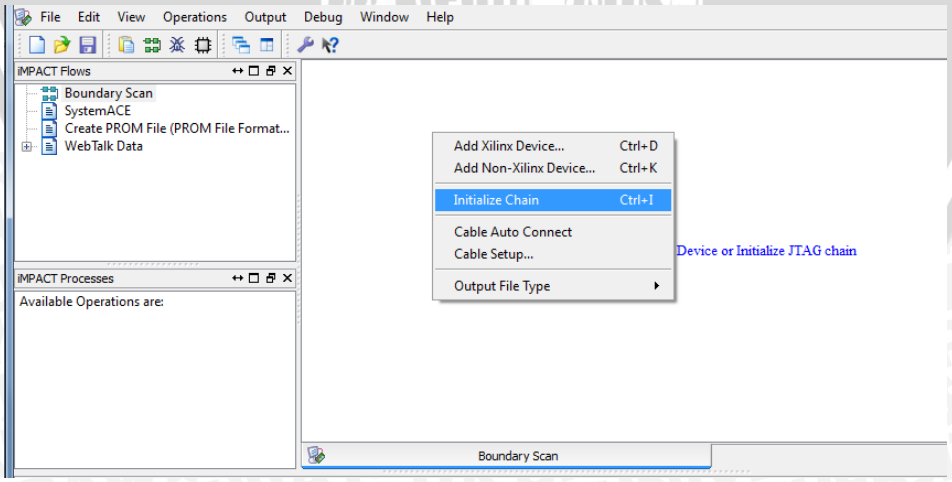
Gambar 4. 3 Menghubungkan Laptop dengan FPGA

Setelah laptop dan FPGA dihubungkan dengan kabel USB langkah selanjutnya adalah *configure target service* hal ini dilakukan untuk mengidentifikasi target import program yang dalam hal ini digunakan FPGA Spartan 3e. Pada saat klik *configure target service* akan keluar ISE impact pada modul ini akan digunakan untuk mengidentifikasi FPGA Spartan 3e. Gambar 4.4 menunjukan proses *configure target service.*



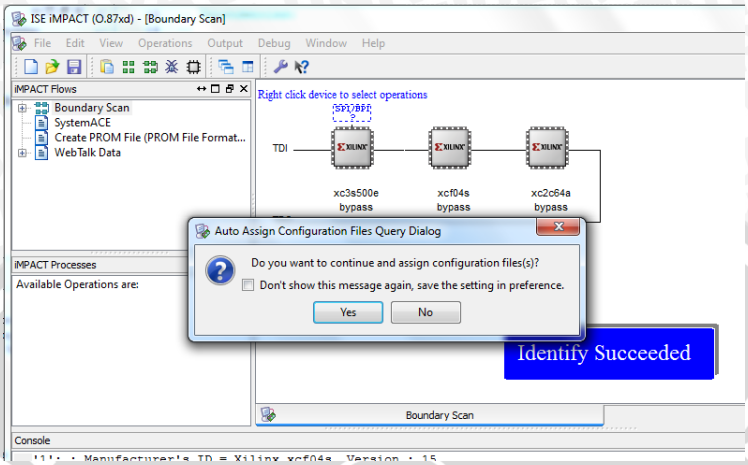
Gambar 4. 4 Configure Target Service

Langkah selanjutnya pilih boundary scan pada ISE Impact setelah itu klik kanan pada ISE Impact dan pilih initialize chain hal ini untuk identifikasi IC yang akan dipilih seperti yang ditunjukan pada Gambar 4.5.



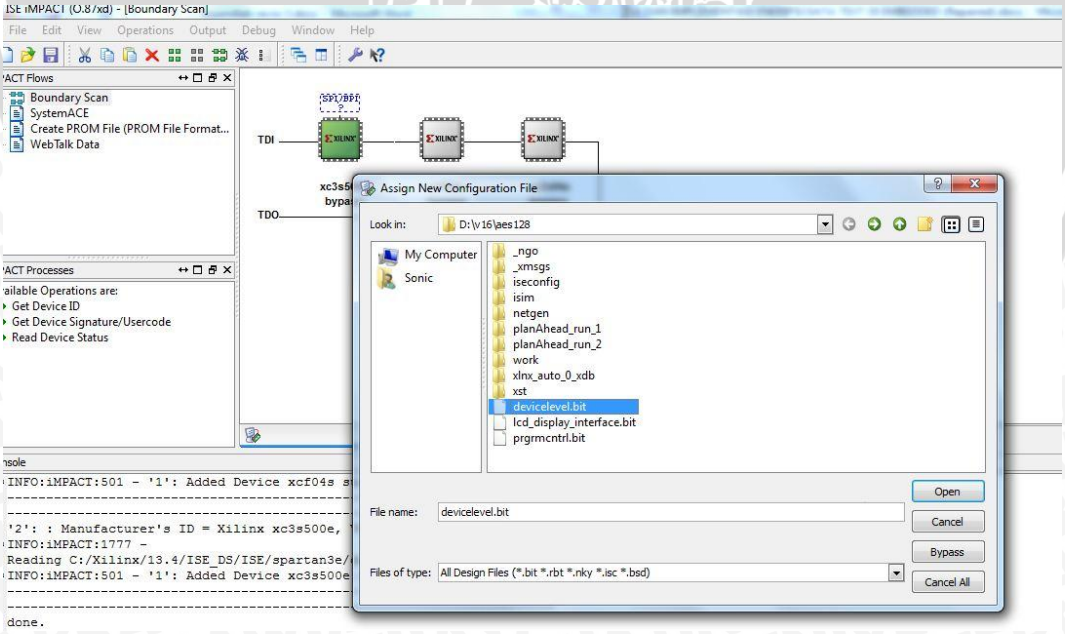
Gambar 4. 5 Initialize Chain

Setelah pilih initialize chain akan keluar pilihan IC yang akan digunakan dalam FPGA Spartan 3e dan pilihan yes untuk langkah selanjutnya seperti yang ditunjukan pada Gambar 4.6



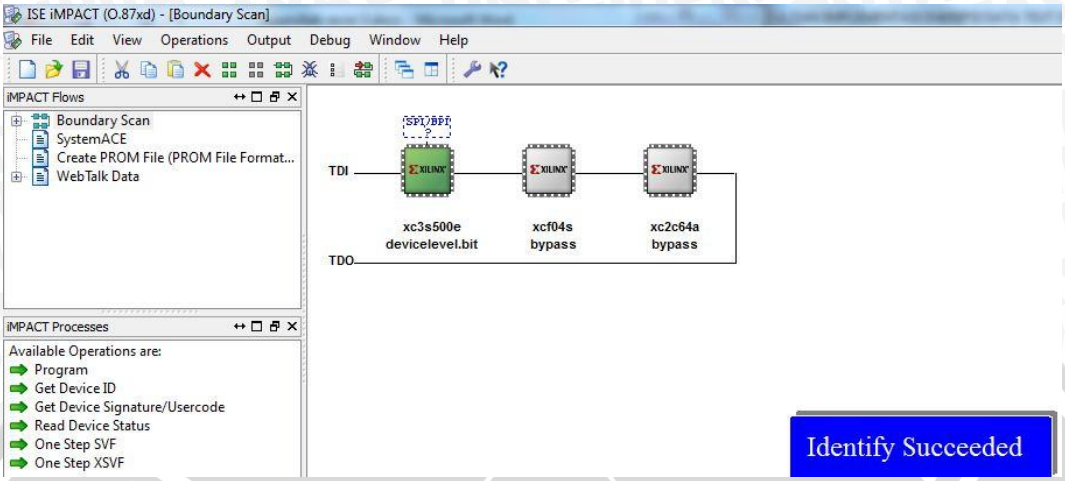
Gambar 4. 6 Identifikasi IC

Setelah pilih yes akan dengan sendirinya mengara pada IC xc3s500s yang akan kita gunakan untuk mengimport program. Langkah selanjutnya adalah pilih dimana file program disimpan yang dalam penelitian ini program bernama devicelevel.bit dan klik open. Proses pemilihan program ditunjukan dalam Gambar 4.7



Gambar 4. 7 Import Program

Setelah program berhasil diimportkan pada IC xc3s500s akan berganti nama dengan nama program yang kita importkan yang dalam penelitian ini bernama devicelevel.bit seperti yang ditunjukan dalam Gambar 4.8.



Gambar 4. 8 Import Program Berhasil

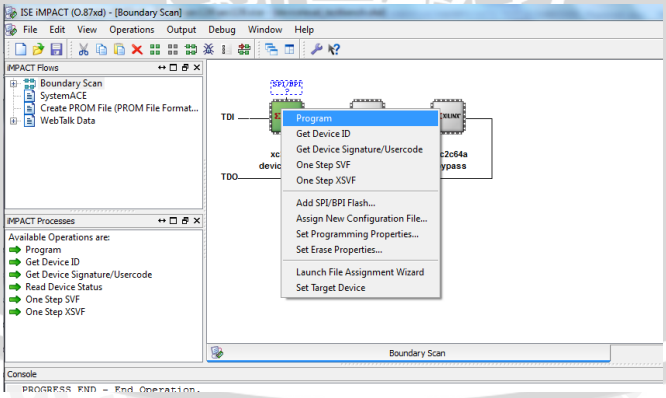
Pada langkah ini merupakan langkah terakhir dari proses import program dalam FPGA dan dalam proses ini program sudah berhasil diimportkan di dalam FPGA.

4.4.2 Menampilkan Program

Langkah pertama adalah menghubungkan antara laptop dengan kabel USB dan menghubungkan keyboard PS2 pada FPGA Spartan 3e seperti yang ditunjukan dalam Gambar 4.9

Gambar 4. 9 Menghubungkan Laptop FPGA dan Keyboard PS2

Setelah laptop FPGA dan Keyboard PS2 berhasil dihubungkan langkah selanjutnya adalah, klik kanan pada IC xcs500s yang digunakan dan pilih program seperti yang ditunjukan dalam Gambar 4.10



Gambar 4. 10 Tampilkan Program

Setelah dilakukan proses pada Gambar 4.10 dengan sendirinya hasil program yang kita rancang akan ditampilkan dalam FPGA Spartan 3e seperti yang ditunjukan pada Gambar 4.11

Gambar 4. 11 Program Sukses Ditampilkan

BAB V
PENGUJIAN DAN ANALISIS

5.1 Skenario Pengujian

Skenario pengujian terhadap enkripsi data text dengan algoritma AES pada FPGA berdasarkan pada BAB III. Pengujian dilakukan dengan 3 tahap yaitu, pengujian validitas, pengujian fungsional dan pengujian waktu. Pengujian yang pertama adalah pengujian fungsional. Pengujian ini dilakukan untuk mengetahui masukan dan hasil keluaran itu selaras. Pengujian ini dilakukan dengan cara membandingkan inputan dan output yang dikeluarkan.

Pengujian kedua adalah pengujian validitas hasil Enkripsi system. Pengujian dilakukan dengan cara membandingkan hasil cipherteks dari system yang dibuat dengan hasil cipherteks yang dihasilkan oleh kalkulator AES dengan kunci yang sama. Jika memang sistem memberikan hasil yang valid maka seharusnya hasil yang dikeluarkan maka akan sama dengan hasil yang dikeluarkan oleh kalkulator AES. Dari pengujian ini akan diambil 10 Data yang akan di uji.

Pengujian ketiga adalah pengujian waktu. Pengujian ini dilakukan untuk mengetahui waktu eksekusi yang dibutuhkan sistem untuk mengenkripsi data. Pengujian ini akan dilakukan dengan cara memasukkan beberapa sampel data yang berbeda panjang karakternya. Hasil dari setiap panjang karakter akan dibandingkan dan di analisa.

5.2 Hasil Pengujian

5.2.1 Pengujian Fungsional

Pengujian fungsional dilakukan dengan cara memeriksa inputan yang dilakukan user pada system. Tujuan dari pengujian ini adalah untuk memastikan inputan yang dilakukan user diterima dengan baik oleh system. Hasil dari pengujian ini akan ditunjukkan pada Tabel 5.1

Tabel 5. 1 Hasil pengujian fungsional

no	Input	Hasil yang diharapkan	Ket
1	tombol a	a	benar
2	tombol b	b	benar
3	tombol c	c	benar
4	tombol d	d	benar
5	tombol e	e	benar
6	tombol f	f	benar
7	tombol g	g	benar
8	tombol h	h	benar
9	tombol i	i	benar
10	tombol j	j	benar
11	tombol k	k	benar
12	tombol l	l	benar
13	tombol m	m	benar
14	tombol n	n	benar
15	tombol o	o	benar
16	tombol p	p	benar
17	tombol q	q	benar
18	tombol r	r	benar
19	tombol s	s	benar
20	tombol t	t	benar
21	tombol u	u	benar
22	tombol v	v	benar
23	tombol w	w	benar
24	tombol x	x	benar
25	tombol y	y	benar
26	tombol z	z	benar
27	shift + a	A	benar
28	shift + b	B	benar
29	shift + c	C	benar

30	shift + d	D	benar
31	shift + e	E	benar
32	shift + f	F	benar
33	shift + g	G	benar
34	shift + h	H	benar
35	shift + i	I	benar
36	shift + j	J	benar
37	shift + k	K	benar
38	shift + l	L	benar
39	shift + m	M	benar
40	shift + n	N	benar
41	shift + o	O	benar
42	shift + p	P	benar
43	shift + q	Q	benar
44	shift + r	R	benar
45	shift + s	S	benar
46	shift + t	T	benar
47	shift + u	U	benar
48	shift + v	V	benar
49	shift + w	W	benar
50	shift + x	X	benar
51	shift + y	Y	benar
52	shift + z	Z	benar
53	Esc	plain teks menjadi hex	benar
54	F1	melakukan enkripsi	benar
55	Switch Reset	Reset Program	benar

Dari hasil pengujian fungsional yang dilakukan di dapatkan hasil semua tombol inputan sesuai dengan yang diharapkan dan tidak terjadi satu kesalahanpun.

5.2.2 Pengujian Validitas

Pengujian ini akan memakai 10 data acak yang hasilnya akan dibandingkan dengan kalkulator AES. Hasil dari pengujian ini akan di perlihatkan pada tabel 5.2

Tabel 5. 2 Hasil pengujian validitas

No	plain teks	cipher teks	ket
1	sawung	aabcdcd8321ca98156d793965c1135b6	benar
2	hello world	cf05981396918d8e162276b65a90266a	benar
3	pamungkas	7c7be804ef8a2b4ec9a804bde0b5e625	benar
4	brawijaya	3bf18517fa5adf281d07c5aeb7bb0f2c	benar
5	universitas	5467d7aec14e2b2af0bcd88ba781db56	benar
6	robotika	5e1485e3ddc18cbb45d1851c2f26622f	benar
7	skripsi	c75420162a6038071f2103ddad89e6de	benar
8	berhasil	23a5dddf9107d892958f1ebceb4fcb40	benar
9	amin	9eecbb879cea0c1c29ce9098e0055cc5	benar
10	prameswari	39e46f66a7a363c65a0dabb6ab4b6c2d	benar

Dari data diatas ternyata dari 10 data mendapatkan hasil yang benar semua. Dari data diatas akan diambil satu data yang akan dibuktikan kebenarannya.

Plain teks : prameswari

Plain teks dalam hex : 7072616D65737761726920202020202020

Cipher teks : 39e46f66a7a363c65a0dabb6ab4b6c2d

Dengan key : 000102030405060708090a0b0c0d0e0f

R0 (Key = 000102030405060708090a0b0c0d0e0f) = 7073636e617671667a602a2b2c2d2e2f

R1 (Key = d6aa74fdd2af72fadaa678f1d6ab76fe) = ccaaf3f9d2379514cef2c2c1ecd0ddb8

R2 (Key = b692cf0b643dbdf1be9bc5006830b3fe) = dcf5e0b1cd40a854d41d311452bb0dc1

R3 (Key = b6ff744ed2c2c9bf6c590cbf0469bf41) = 0541e6e15b86269b18c6172c0d19d38a

R4 (Key = 47f7f7bc95353e03f96c32bcfd058dfd) = d361a149be2d6c7145ef702250bcbf09

R5 (Key = 3caaa3e8a99f9deb50f3af57adf622aa) = d395bcc9a7b7b1a3b2a23eefe20390dd

R6 (Key = 5e390f7df7a69296a7553dc10aa31f6b) = 011ae755bcf348892b31a59248f35c9f

R7 (Key = 14f9701ae35fe28c440adf4d4ea9c026) = 264e7b38cd8dcb4d99e1e2890a1d45be

R8 (Key = 47438735a41c65b9e016baf4aebf7ad2) = 6313efb5bffff5fd122b8c85a900fe1e

R9 (Key = 549932d1f08557681093ed9cbe2c974e) = 959081518677a78bb7b21ec8f5a34c8e

R10 (Key = 13111d7fe3944a17f307a78b4d2b30c5) = 39e46f66a7a363c65a0dabb6ab4b6c2d

Hasil akhir = 39e46f66a7a363c65a0dabb6ab4b6c2d

Dari plainteks prameswari akan dijadikan dalam bentuk hex terlebih dahulu, setelah itu akan di proses dengan kunci yang sudah ada. Dalam setiap round akan mendapatkan kunci baru lagi. Dalam setiap round juga sudah dilakukan transformasi AES, antara lainnya SubByte, ShiftRows dan MixColumns. Pada akhirnya akan mendapatkah hasil "39e46f66a7a363c65a0dabb6ab4b6c2d" yang dimana hasil tersebut masih dalam bentuk hex. Hasil enkripsi pada FPGA akan ditunjukkan pada Gambar 5.1 dan 5.2.

Gambar 5. 1 Teks sebelum di enkripsi (plainteks)

Gambar 5. 2 Gambar setelah dienkripsi (CipherTeks)

Dari pengujian validitas yang telah dilakukan maka dapat disimpulkan enkripsi yang berjalan pada FPGA berjalan dengan baik dan sesuai harapan.

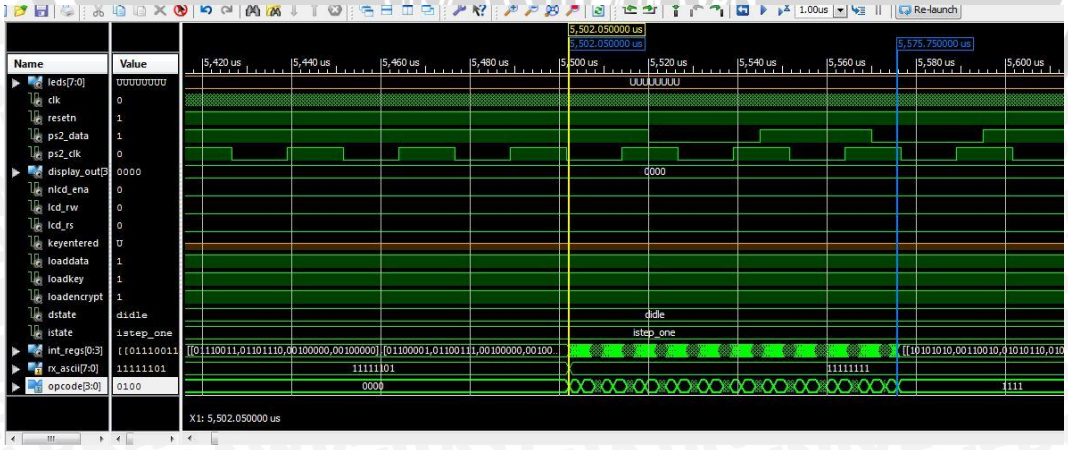
5.2.3 Pengujian Waktu

Pengujian waktu akan dilakukan dengan simulator pada ISE Project Navigator. Dari sana akan didapatkan waktu yang dibutuhkan. Dalam kasus ini dimasukkan 10 data yang sama dengan data yang dimasukkan pada pengujian validitas. Hasil dari pengujian Waktu akan ditunjukkan pada Tabel 5.3

Tabel 5. 3 Hasil Pengujian Waktu

NO	PLAIN TEKS	WAKTU
1	sawung	73.700000 us
2	hello world	73.700000 us
3	pamungkas	73.700000 us
4	brawijaya	73.700000 us
5	universitas	73.700000 us
6	robotika	73.700000 us
7	skripsi	73.700000 us
8	berhasil	73.700000 us
9	amin	73.700000 us
10	prameswari	73.700000 us

Pengujian yang dilakukan pada simulator diambil dari panjang waktu opcode pada system yang menunjukkan proses jalan enkripsi yang dilakukan. Hasil pengujian pada salah satu system akan ditunjukkan pada Gambar 5.3



Gambar 5. 3 Gambar simulator pada pengujian waktu

Pada gambar 5.3 dilakukan pengujian dengan memasukkan data “sawung” pada program. Pada proses berjalan pada waktu 5,502.050000 us – 5,575.750000 us, dari hasil tersebut maka diperoleh waktu 73.700000 us. Dari hasil pengujian ini mendapatkan data bahwa panjang pendeknya karakter pada enkripsi AES 128 bit tidak mempengaruhi waktu yang dibutuhkan selama panjang karakter tersebut masih dalam 1 blok yaitu 16 karakter atau 16 bytes.

BAB VI
PENUTUP

6.1 Kesimpulan

Berdasarkan hasil perancangan, implementasi dan pengujian yang dilakukan, maka diambil kesimpulan sebagai berikut :

1. Implementasi enkripsi data teks di *embedded system* dengan metode AES dapat diimplementasikan pada FPGA Spartan 3e. FPGA Spartan 3e digunakan sebagai simulator pemrograman hardware.
2. Berdasarkan hasil pengujian yang dilakukan pada pengujian pertama yaitu pengujian validitas dan pengujian fungsional yaitu pengujian fungsional, keduanya menghasilkan hasil yang sesuai harapan. Maka status validitasnya dikatakan valid.
3. Pengujian validitas dan fungsional memberikan hasil 100% system benar dalam menghasilkan enkripsi suatu data teks.
4. Berdasarkan hasil pengujian waktu, panjang karakter tidak mempengaruhi waktu yang digunakan untuk mengenkripsi data teks selama panjang karakter itu tidak melebihi 1 panjang blok yaitu 16 karakter atau 16 bytes.
5. Pengujian waktu memberikan hasil waktu yang diperlukan untuk mengenkripsi data teks di FPGA adalah 73.700000 us.

6.2 Saran

Saran yang dapat diberikan untuk pengembangan penelitian ini antara lain:

1. Untuk pengembangan lebih lanjut, diharapkan diciptakan juga untuk dekripsi data teks.
2. Untuk pengembangan lebih lanjut, diharapkan sudah bisa di implementasikan di dalam IC yang sebenarnya agar enkripsi berbasis *embedded system* segera tercipta.
3. Untuk pengujian lebih lanjut, bisa dilakukan pengujian serangan seperti *exhaustive attack* atau *brute force.*

DAFTAR PUSTAKA

[Maman-2012] Abdurohman, Maman. 2012 "*Perancangan Embedded System Berbasis FPGA"* Graha Ilmu, Bandung

[Rifki-2012] Sadikin, Rifki. 2012 "*Kriptografi untuk keamanan jaringan"* Penerbit Andi, Yogyakarta

[Padmaja-2010] DR. V. PADMAJA 2010. "*Secure Embedded System Networking: An Advanced Security Perspective*". Department of Electronics and Communication Engineering,VNR VJIET, Hyderabad, India

[Srinidhi-2011] Kestur, Srinidhi. D.Davis, John. Williams, Oliver, 2011. "*BLAS Comparison on FPGA, CPU and GPU*".Dept of Computer Science and Engineering The Pennyslvania State University.

[Bernandino-2010] Adiwijaya, Bernardino M D. "*Algoritma AES (Advanced Encryption Standard) dan Penggunaannya dalam Penyandian Pengompresian Data*". Program Studi Teknik Informatika, Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung. Bandung

[Kunjung-2010] Wahyudi, Kunjung, DP.Silitonga, Parasian. 2010 "*Aplikasi Kriptografi untuk Pertukaran Pesan Menggunakan Teknik Stenaganografi dan Algoritma AES"* Jurusan Teknik Informatika Fakultas Teknologi Informasi, Institut Teknologi Adhi Tama Surabaya. Surabaya.

[Sugi-2010] Guritman, Sugi. Ridha, Ahmad. Purnama Giri, Endang. 2010 "*ANALISIS ALGORITME DAN WAKTU ENKRIPSI VERSUS DEKRIPSI PADA ADVANCED ENCRYPTION STANDARD (AES)"*.

[Benaissa-2009] Benaissa, Mohammed. 2009 "*AES on FPGA from the fastest to the smallest"*.

[Samanta-2009] Samanta, Sounak. 2009 "*FPGA Implementation of AES Encryption and Decryption"*

[Kaur-2013] Kaur, Amandeep. 2013 "*FPGA Implementation of Efficient Hardware for the Advanced Encryption Standard"*

LAMPIRAN SOURCE CODE PROGRAM

A. SourceCode SubByte

```
when "0101" => --subbytes
        status<="100";
                MAIN_counter<=MAIN_counter+1;
                case MAIN_counter is
                when 0 =>
                 MAIN_counter<=1;
                 address<= "100" & INT_REGS(0,0); --word 0
                when 1 =>
                 INT_REGS(0,0)<=rombus;
                when 2 =>
                 address<="100" & INT_REGS(1,0);
                when 3 =>
                 INT_REGS(1,0)<=rombus;
                when 4 =>
                 address<="100"& INT_REGS(2,0);
                when 5 =>
                 INT_REGS(2,0)<=rombus;
                when 6 =>
                 address<="100"& INT_REGS(3,0);
                when 7 =>
                 INT_REGS(3,0)<=rombus;
                when 8 =>
                 address<="100"& INT_REGS(0,1); --word 1
                when 9 =>
                 INT_REGS(0,1)<=rombus;
                when 10 =>
                 address<="100" & INT_REGS(1,1);
                when 11 =>
                 INT_REGS(1,1)<=rombus;
                when 12 =>
                 address<="100" & INT_REGS(2,1);
                when 13 =>
                 INT_REGS(2,1)<=rombus;
                when 14 =>
                 address<="100"& INT_REGS(3,1);
                when 15 =>
                 INT_REGS(3,1)<=rombus;
                when 16 =>
                 address<="100"& INT_REGS(0,2); --word 2
                when 17 =>
                 INT_REGS(0,2)<=rombus;
```

```
42.              when 18 =>
43.               address<="100"& INT_REGS(1,2);
44.              when 19 =>
45.               INT_REGS(1,2)<=rombus;
46.              when 20 =>
47.               address<="100" & INT_REGS(2,2);
48.              when 21 =>
49.               INT_REGS(2,2)<=rombus;
50.              when 22 =>
51.               address<="100"& INT_REGS(3,2);
52.              when 23 =>
53.               INT_REGS(3,2)<=rombus;
54.              when 24 =>
55.               address<="100"& INT_REGS(0,3); --word 3
56.              when 25 =>
57.               INT_REGS(0,3)<=rombus;
58.              when 26 =>
59.               address<="100"& INT_REGS(1,3);
60.              when 27 =>
61.               INT_REGS(1,3)<=rombus;
62.              when 28 =>
63.               address<="100"& INT_REGS(2,3);
64.              when 29 =>
65.               INT_REGS(2,3)<=rombus;
66.              when 30 =>
67.               address<="100"& INT_REGS(3,3);
68.              when 31 =>
69.               INT_REGS(3,3)<=rombus;
70.              when 32=>
71.               MAIN_counter<=0;
72.               status<="010";
73.      when others =>
74.       status<="101";
75.      end case;
```

B. SourceCode ShiftRows

```
1.  when "0110" =>  --shift rows
2.        status<="100";
3.                MAIN_counter<=MAIN_counter+1;
4.        case MAIN_counter is
5.        when 0 =>
6.         tempbyte1<=INT_REGS(1,0);
7.                        tempbyte2<=INT_REGS(2,0);
8.                        tempbyte3<=INT_REGS(3,0);
9.                        tempbyte4<=INT_REGS(2,1);
10.                       tempbyte5<=INT_REGS(1,2);
```

```
11.                     tempbyte6<=INT_REGS(3,2);
12.       when 1=>
13.         INT_REGS(1,0)<=INT_REGS(1,1);
14.         INT_REGS(2,0)<=INT_REGS(2,2);
15.         INT_REGS(3,0)<=INT_REGS(3,3);
16.         INT_REGS(2,1)<=INT_REGS(2,3);
17.         INT_REGS(1,2)<=INT_REGS(1,3);
18.                     INT_REGS(3,2)<=INT_REGS(3,1);
19.       when 2=>
20.         INT_REGS(1,3)<=tempbyte1;
21.                     INT_REGS(2,2)<=tempbyte2;
22.         INT_REGS(3,1)<=tempbyte3;
23.         INT_REGS(2,3)<=tempbyte4;
24.         INT_REGS(1,1)<=tempbyte5;
25.         INT_REGS(3,3)<=tempbyte6;
26.       when 3 =>
27.        MAIN_counter<=0;
28.        status<="010";
29.       when others =>
30.        status<="101";
31.       end case;
```

C. SourceCode MixColumns

```
1.  when "0111" =>  -- MixColumns
2.       status<="100";
3.               MAIN_counter<=MAIN_counter+1;
4.       case MAIN_counter is
5.       when 0 =>
6.          --calc (0,0)
7.                  inv0(8 downto 0):=INT_REGS(0,0) & '0' xor
8.                  INT_REGS(1,0) & '0' xor
9.                  '0' & INT_REGS(1,0) xor
10.                 '0' & INT_REGS(2,0) xor
11.                 '0' & INT_REGS(3,0);
12.
13.                 --calc(1,0)
14.                 inv1(8 downto 0):=INT_REGS(1,0) & '0' xor
15.                 INT_REGS(2,0) & '0' xor
16.                 '0' & INT_REGS(2,0) xor
17.                 '0' & INT_REGS(0,0) xor
18.                 '0' & INT_REGS(3,0);
19.
20.                 --calc(2,0)
21.                 inv2(8 downto 0):=INT_REGS(2,0) & '0' xor
22.                 INT_REGS(3,0) & '0' xor
23.                 '0' & INT_REGS(3,0) xor
```

```
                '0' & INT_REGS(0,0) xor
                '0' & INT_REGS(1,0);

                --calc(3,0)
                inv3(8 downto 0):=INT_REGS(3,0) & '0' xor
                INT_REGS(0,0) & '0' xor
                '0' & INT_REGS(0,0) xor
                '0' & INT_REGS(1,0) xor
                '0' & INT_REGS(2,0);

               if(inv0(8)='1') then
                     inv0(7 downto 0):=inv0(7 downto 0) xor
   "00011011";
        end if;
                if(inv1(8)='1') then
                     inv1(7 downto 0):=inv1(7 downto 0) xor
   "00011011";
        end if;
                if(inv2(8)='1') then
                     inv2(7 downto 0):=inv2(7 downto 0) xor
   "00011011";
        end if;
                if(inv3(8)='1') then
                     inv3(7 downto 0):=inv3(7 downto 0) xor
   "00011011";
        end if;

                    INT_REGS(0,0)<=inv0(7 downto 0);
                    INT_REGS(1,0)<=inv1(7 downto 0);
                    INT_REGS(2,0)<=inv2(7 downto 0);
                    INT_REGS(3,0)<=inv3(7 downto 0);

                    MAIN_counter<=1;
     when 1 =>
        --calc (0,1)
                inv0(8 downto 0):=INT_REGS(0,1) & '0' xor
                INT_REGS(1,1) & '0' xor
                '0' & INT_REGS(1,1) xor
                '0' & INT_REGS(2,1) xor
                '0' & INT_REGS(3,1);

                --calc(1,1)
                inv1(8 downto 0):=INT_REGS(1,1) & '0' xor
                INT_REGS(2,1) & '0' xor
                '0' & INT_REGS(2,1) xor
                '0' & INT_REGS(0,1) xor
                '0' & INT_REGS(3,1);
```

```
67.
68.                 --calc(2,1)
69.                 inv2(8 downto 0):=INT_REGS(2,1) & '0' xor
70.                 INT_REGS(3,1) & '0' xor
71.                 '0' & INT_REGS(3,1) xor
72.                 '0' & INT_REGS(0,1) xor
73.                 '0' & INT_REGS(1,1);
74.
75.                 --calc(3,1)
76.                 inv3(8 downto 0):=INT_REGS(3,1) & '0' xor
77.                 INT_REGS(0,1) & '0' xor
78.                 '0' & INT_REGS(0,1) xor
79.                 '0' & INT_REGS(1,1) xor
80.                 '0' & INT_REGS(2,1);
81.
82.               if(inv0(8)='1') then
83.                       inv0(7   downto   0):=inv0(7   downto   0)      xor
   "00011011";
84.       end if;
85.               if(inv1(8)='1') then
86.                       inv1(7   downto   0):=inv1(7   downto   0)      xor
   "00011011";
87.       end if;
88.               if(inv2(8)='1') then
89.                       inv2(7   downto   0):=inv2(7   downto   0)      xor
   "00011011";
90.       end if;
91.               if(inv3(8)='1') then
92.                       inv3(7   downto   0):=inv3(7   downto   0)      xor
   "00011011";
93.       end if;
94.
95.                     INT_REGS(0,1)<=inv0(7 downto 0);
96.                     INT_REGS(1,1)<=inv1(7 downto 0);
97.                     INT_REGS(2,1)<=inv2(7 downto 0);
98.                     INT_REGS(3,1)<=inv3(7 downto 0);
99.
100.        when 2 =>
101.           --calc (0,2)
102.                     inv0(8 downto 0):=INT_REGS(0,2) & '0' xor
103.                     INT_REGS(1,2) & '0' xor
104.                     '0' & INT_REGS(1,2) xor
105.                     '0' & INT_REGS(2,2) xor
106.                     '0' & INT_REGS(3,2);
107.
108.                     --calc(1,2)
109.                     inv1(8 downto 0):=INT_REGS(1,2) & '0' xor
```

```
                    INT_REGS(2,2) & '0' xor
                    '0' & INT_REGS(2,2) xor
                    '0' & INT_REGS(0,2) xor
                    '0' & INT_REGS(3,2);

                    --calc(2,2)
                    inv2(8 downto 0):=INT_REGS(2,2) & '0' xor
                    INT_REGS(3,2) & '0' xor
                    '0' & INT_REGS(3,2) xor
                    '0' & INT_REGS(0,2) xor
                    '0' &  INT_REGS(1,2);

                    --calc(3,2)
                    inv3(8 downto 0):=INT_REGS(3,2) & '0' xor
                    INT_REGS(0,2) & '0' xor
                    '0' & INT_REGS(0,2) xor
                    '0' & INT_REGS(1,2) xor
                    '0' & INT_REGS(2,2);

                  if(inv0(8)='1') then
                       inv0(7 downto 0):=inv0(7 downto 0)   xor
    "00011011";
            end if;
                  if(inv1(8)='1') then
                       inv1(7 downto 0):=inv1(7 downto 0)   xor
    "00011011";
            end if;
                  if(inv2(8)='1') then
                       inv2(7 downto 0):=inv2(7 downto 0)   xor
    "00011011";
            end if;
                  if(inv3(8)='1') then
                       inv3(7 downto 0):=inv3(7 downto 0)   xor
    "00011011";
            end if;

                        INT_REGS(0,2)<=inv0(7 downto 0);
                        INT_REGS(1,2)<=inv1(7 downto 0);
                        INT_REGS(2,2)<=inv2(7 downto 0);
                        INT_REGS(3,2)<=inv3(7 downto 0);

        when 3 =>
            --calc (0,3)
                    inv0(8 downto 0):=INT_REGS(0,3) & '0' xor
                    INT_REGS(1,3) & '0' xor
                    '0' & INT_REGS(1,3) xor
                    '0' & INT_REGS(2,3) xor
```

```
153.                    '0' & INT_REGS(3,3);
154.
155.                    --calc(1,3)
156.                    inv1(8 downto 0):=INT_REGS(1,3) & '0' xor
157.                    INT_REGS(2,3) & '0' xor
158.                    '0' & INT_REGS(2,3) xor
159.                    '0' & INT_REGS(0,3) xor
160.                    '0' & INT_REGS(3,3);
161.
162.                    --calc(2,3)
163.                    inv2(8 downto 0):=INT_REGS(2,3) & '0' xor
164.                    INT_REGS(3,3) & '0' xor
165.                    '0' & INT_REGS(3,3) xor
166.                    '0' & INT_REGS(0,3) xor
167.                    '0' & INT_REGS(1,3);
168.
169.                    --calc(3,3)
170.                    inv3(8 downto 0):=INT_REGS(3,3) & '0' xor
171.                    INT_REGS(0,3) & '0' xor
172.                    '0' & INT_REGS(0,3) xor
173.                    '0' & INT_REGS(1,3) xor
174.                    '0' & INT_REGS(2,3);
175.
176.                   if(inv0(8)='1') then
177.                        inv0(7 downto 0):=inv0(7 downto 0)   xor
    "00011011";
178.           end if;
179.                   if(inv1(8)='1') then
180.                        inv1(7 downto 0):=inv1(7 downto 0)   xor
    "00011011";
181.           end if;
182.                   if(inv2(8)='1') then
183.                        inv2(7 downto 0):=inv2(7 downto 0)   xor
    "00011011";
184.           end if;
185.                   if(inv3(8)='1') then
186.                        inv3(7 downto 0):=inv3(7 downto 0)   xor
    "00011011";
187.           end if;
188.
189.                        INT_REGS(0,3)<=inv0(7 downto 0);
190.                        INT_REGS(1,3)<=inv1(7 downto 0);
191.                        INT_REGS(2,3)<=inv2(7 downto 0);
192.                        INT_REGS(3,3)<=inv3(7 downto 0);
193.
194.         when 8 =>
195.           MAIN_counter<=0;
```

```
196.          status<="010";
197.        when others =>
198.          status<="101";
199.       end case;
```

D. SourceCode AddRoundKey

```
1.  when "0100" => -- add round key
2.        status<="100";
3.        MAIN_counter<=MAIN_counter+1;
4.
5.        case MAIN_counter is
6.        when 0 =>
7.         if(inverse='1') then
8.        if(WORD_EXTREGPTR=0) then
9.                WORD_EXTREGPTR<=80;
10.               elsif(WORD_EXTREGPTR=15) then
11.          WORD_EXTREGPTR<=0;
12.               end if;
13.              end if;
14.       when 1 =>
15.              address <= "010" & IntTouVec(WORD_EXTREGPTR,8);
16.       when 2 =>
17.              --ram wait state
18.              tempbyte1<=INT_REGS(0,0);
19.              tempbyte2<=INT_REGS(1,0);
20.       when 3 =>
21.      INT_REGS(0,0)<=rambus(15 downto 8) xor tempbyte1;
22.      INT_REGS(1,0)<=rambus(7 downto 0)  xor tempbyte2;
23.      WORD_EXTREGPTR<=WORD_EXTREGPTR+1;
24.     when 4 =>
25.      address<="010" & IntTouVec(WORD_EXTREGPTR,8);
26.     when 5 =>
27.       --ram wait state
28.      tempbyte1<=INT_REGS(2,0);
29.              tempbyte2<=INT_REGS(3,0);
30.     when 6 =>
31.      INT_REGS(2,0)<=rambus(15 downto 8) xor tempbyte1;
32.      INT_REGS(3,0)<=rambus(7 downto 0) xor tempbyte2;
33.              WORD_EXTREGPTR<=WORD_EXTREGPTR+1;
34.       when 7 =>
35.              address <= "010" & IntTouVec(WORD_EXTREGPTR,8);
36.       when 8 =>
37.        --ram wait state
38.              tempbyte1<=INT_REGS(0,1);
39.              tempbyte2<=INT_REGS(1,1);
40.       when 9 =>
```

```
41.      INT_REGS(0,1)<=rambus(15 downto 8) xor tempbyte1;
42.      INT_REGS(1,1)<=rambus(7 downto 0) xor tempbyte2;
43.      WORD_EXTREGPTR<=WORD_EXTREGPTR+1;
44.     when 10 =>
45.       address<="010" & IntTouVec(WORD_EXTREGPTR,8);
46.    when 11 =>
47.      --ram wait state
48.               tempbyte1<=INT_REGS(2,1);
49.               tempbyte2<=INT_REGS(3,1);
50.    when 12 =>
51.      INT_REGS(2,1)<=rambus(15 downto 8) xor tempbyte1;
52.      INT_REGS(3,1)<=rambus(7 downto 0) xor tempbyte2;
53.               WORD_EXTREGPTR<=WORD_EXTREGPTR+1;
54.      when 13 =>
55.               address <= "010" & IntTouVec(WORD_EXTREGPTR,8);
56.      when 14 =>
57.               --ram wait state
58.               tempbyte1<=INT_REGS(0,2);
59.               tempbyte2<=INT_REGS(1,2);
60.      when 15 =>
61.      INT_REGS(0,2)<=rambus(15 downto 8) xor tempbyte1;
62.      INT_REGS(1,2)<=rambus(7 downto 0) xor tempbyte2;
63.      WORD_EXTREGPTR<=WORD_EXTREGPTR+1;
64.     when 16 =>
65.      address<="010" & IntTouVec(WORD_EXTREGPTR,8);
66.     when 17 =>
67.      --ram wait state
68.               tempbyte1<=INT_REGS(2,2);
69.               tempbyte2<=INT_REGS(3,2);
70.     when 18 =>
71.       INT_REGS(2,2)<=rambus(15 downto 8) xor tempbyte1;
72.       INT_REGS(3,2)<=rambus(7 downto 0) xor tempbyte2;
73.                WORD_EXTREGPTR<=WORD_EXTREGPTR+1;
74.              when 19 =>
75.                address <= "010" &
    IntTouVec(WORD_EXTREGPTR,8);
76.              when 20 =>
77.               --ram wait state
78.                     tempbyte1<=INT_REGS(0,3);
79.                     tempbyte2<=INT_REGS(1,3);
80.              when 21 =>
81.       INT_REGS(0,3)<=rambus(15 downto 8) xor tempbyte1;
82.       INT_REGS(1,3)<=rambus(7 downto 0) xor tempbyte2;
83.      WORD_EXTREGPTR<=WORD_EXTREGPTR+1;
84.     when 22 =>
85.      address<="010" & IntTouVec(WORD_EXTREGPTR,8);
86.      when 23 =>
```

```
87.         --ram wait state
88.                         tempbyte1<=INT_REGS(2,3);
89.                 tempbyte2<=INT_REGS(3,3);
90.       when 24 =>
91.         INT_REGS(2,3)<=rambus(15 downto 8) xor tempbyte1;
92.         INT_REGS(3,3)<=rambus(7 downto 0) xor tempbyte2;

93.                 when 25=>
94.                  MAIN_counter<=0;
95.                  if(inverse='0') then
96.                    if(WORD_EXTREGPTR<87) then
97.                      WORD_EXTREGPTR<=WORD_EXTREGPTR+1;
98.                    else
99.                     WORD_EXTREGPTR<=0;
100.                     end if;
101.                   else
102.                     if(WORD_EXTREGPTR>16) then
103.                       WORD_EXTREGPTR<=WORD_EXTREGPTR-
    15;
104.                     elsif(WORD_EXTREGPTR=15) then
105.                     --indicate not to reset to 80 at next call
106.                     else
107.                       WORD_EXTREGPTR<=0;
108.                     end if;
109.                   end if;
110.                   status<="010";
111.                  when others =>
112.                   rambus<=(others =>'Z');
113.           status<="101";
114.              end case;
```

E. Keyboard_PS2

```
1.  begin
2.  reset<=not resetn;
3.  ps2_direction : process( ps2_clk_hi_z, ps2_data_hi_z )
4.  begin
5.    if( ps2_clk_hi_z = '1' ) then
6.       ps2_clk <= 'Z';
7.    else
8.       ps2_clk <= '0';
9.    end if;
10.  if( ps2_data_hi_z = '1' ) then
11.     ps2_data <= 'Z';
12.  else
13.     ps2_data <= '0';
14.  end if;
```

```
15. end process;
16. ps2_synch : process(clk, ps2_clk, ps2_data)
17. begin
18.   if clk'event and clk='0' then
19.     ps2_clk_s <= ps2_clk;
20.     ps2_data_s <= ps2_data;
21.   end if;
22. end process;
23.
24. -- State register
25. m1_state_register : process( clk, reset, m1_state )
26. begin
27.   if clk'event and clk='0' then
28.     if (reset = '1') then
29.         m1_state <= m1_rx_clk_h;
30.     else
31.         m1_state <= m1_next_state;
32.     end if;
33.   end if;
34. end process;
35.
36. m1_state_logic : process( m1_state, q,
37.                                         ps2_clk_s, ps2_data_s,
38.                                         timer_60usec_done,
    timer_5usec_done )
39. begin
40.   -- Output signals default to this value, unless changed in a state
    condition.
41.   ps2_clk_hi_z          <= '1';
42.   ps2_data_hi_z         <= '1';
43.   enable_timer_60usec     <= '0';
44.   enable_timer_5usec      <= '0';
45.
46.   case (m1_state) is
47.   when m1_rx_clk_h =>
48.       enable_timer_60usec <= '1';
49.       if (ps2_clk_s = '0') then
50.                 m1_next_state <= m1_rx_falling_edge_marker;
51.       else
52.                 m1_next_state <= m1_rx_clk_h;
53.       end if;
54.
55.   when m1_rx_falling_edge_marker =>
56.       enable_timer_60usec <= '0';
57.       m1_next_state <= m1_rx_clk_l;
58.
59.   when m1_rx_clk_l =>
```

```
60.       enable_timer_60usec <= '1';
61.       if (ps2_clk_s = '1') then
62.                 m1_next_state <= m1_rx_rising_edge_marker;
63.       else
64.                 m1_next_state <= m1_rx_clk_l;
65.       end if;
66.
67.   when m1_rx_rising_edge_marker =>
68.       enable_timer_60usec <= '0';
69.       m1_next_state <= m1_rx_clk_h;
70.   when others =>
71.         m1_next_state <= m1_rx_clk_h;
72.   end case;
73. end process;
74.
75. -- This is the bit counter
76. bit_counter: process(clk, reset, m1_state, bit_count )
77. begin
78.   if clk'event and clk = '0' then
79.     if ( reset = '1' ) or
80.       ( rx_shifting_done = '1' ) then
81.       bit_count <= "0000";  -- normal reset
82.     elsif (timer_60usec_done = '1' ) and
83.         (m1_state = m1_rx_clk_h)        and
84.         (ps2_clk_s = '1') then
85.       bit_count <= "0000";  -- rx watchdog timer reset
86.     elsif (m1_state = m1_rx_falling_edge_marker)then   -- increment for
    rx
87.       bit_count <= bit_count + 1;
88.     end if;
89.   end if;
90. end process;
91.
92. assign: process( bit_count, m1_state )
93. begin
94.   if (bit_count = TOTAL_BITS) then
95.     rx_shifting_done <= '1';
96.   else
97.     rx_shifting_done <= '0';
98.   end if;
99. end process;
100.
101.    -- This is the shift register
102.    q_shift : process(clk, m1_state, q, ps2_data_s, rx_shifting_done )
103.    begin
104.      if clk'event and clk='0' then
105.        if (reset = '1') then
```

```
106.                q <= "00000000000";
107.        elsif ( (m1_state = m1_rx_falling_edge_marker) ) then
108.          q <= ps2_data_s & q((TOTAL_BITS-1) downto 1);
109.        end if;
110.      end if;
111.      if (q(8 downto 1) = EXTEND_CODE) and (rx_shifting_done = '1')
    then
112.        extended <= '1';
113.      else
114.        extended <= '0';
115.      end if;
116.      if (q(8 downto 1) = RELEASE_CODE) and (rx_shifting_done =
    '1') then
117.        released <= '1';
118.      else
119.        released <= '0';
120.      end if;
121.    end process;
122.
123.    -- This is the 60usec timer counter
124.    timer60usec:          process(clk,          enable_timer_60usec,
    timer_60usec_count)
125.    begin
126.      if clk'event and clk = '0' then
127.        if (enable_timer_60usec = '0') then
128.                timer_60usec_count <= "000000000000";
129.        elsif (timer_60usec_done = '0') then
130.                timer_60usec_count <= timer_60usec_count + 1;
131.        end if;
132.      end if;
133.
134.      if (timer_60usec_count = (TIMER_60USEC_VALUE_PP - 1)) then
135.              timer_60usec_done <= '1';
136.      else
137.        timer_60usec_done <= '0';
138.      end if;
139.    end process;
140.
141.    -- This is the 5usec timer counter
142.    timer5usec : process(clk, enable_timer_5usec, timer_5usec_count )
143.    begin
144.      if clk'event and clk = '0' then
145.        if (enable_timer_5usec = '0') then
146.                timer_5usec_count <= "00000000";
147.        elsif (timer_5usec_done = '0') then
148.                timer_5usec_count <= timer_5usec_count + 1;
149.        end if;
```

```
150.      end if;
151.
152.      if( timer_5usec_count = (TIMER_5USEC_VALUE_PP - 1)) then
153.              timer_5usec_done <= '1';
154.      else
155.              timer_5usec_done <= '0';
156.      end if;
157.    end process;
158.    special_scan : process(clk, reset, rx_output_event, rx_shifting_done,
      extended, released )
159.    begin
160.      if clk'event and clk='0' then
161.        if (reset = '1') or (rx_output_event = '1')then
162.          hold_extended <= '0';
163.          hold_released <= '0';
164.        else
165.          if (rx_shifting_done = '1') and (extended = '1') then
166.                hold_extended <= '1';
167.          end if;
168.          if (rx_shifting_done = '1') and (released = '1') then
169.                hold_released <= '1';
170.          end if;
171.        end  if;
172.      end if;
173.    end process;
174.
175.    -- These bits contain the status of the two shift keys
176.    left_shift_proc     :     process(clk,     reset,     q,     rx_shifting_done,
      hold_released )
177.    begin
178.      if clk'event and clk = '0' then
179.        if (reset = '1') then
180.              left_shift_key <= '0';
181.        elsif (q(8 downto 1) = LEFT_SHIFT) and
182.                (rx_shifting_done = '1') and
183.                        (hold_released = '0') then
184.          left_shift_key <= '1';
185.        elsif (q(8 downto 1) = LEFT_SHIFT) and
186.                (rx_shifting_done = '1') and
187.                        (hold_released = '1') then
188.          left_shift_key <= '0';
189.        end if;
190.      end if;
191.    end process;
192.
193.    right_shift_proc     :     process(clk,     reset,     q,     rx_shifting_done,
      hold_released )
```

```
194.    begin
195.     if clk'event and clk = '0' then
196.      if (reset = '1') then
197.      right_shift_key <= '0';
198.      elsif (q(8 downto 1) = RIGHT_SHIFT) and
199.                  (rx_shifting_done = '1') and
200.                              (hold_released = '0') then
201.        right_shift_key <= '1';
202.      elsif (q(8 downto 1) = RIGHT_SHIFT) and
203.                  (rx_shifting_done = '1') and
204.                              (hold_released = '1') then
205.        right_shift_key <= '0';
206.      end if;
207.     end if;
208.    end process;
209.
210.    shift_key_on <= left_shift_key or right_shift_key;
211.    special_scan_proc : process(clk, reset,
212.                                                    hold_extended,
    hold_released,
213.                                                    q,          ascii,
    ctrl_key_on )
214.    begin
215.     if clk'event and clk = '0' then
216.      if (reset = '1')      then
217.        rx_extended <= '0';
218.        rx_released <= '0';
219.     --     rx_scan_code <= "00000000";
220.        rx_ascii <= "11111111";
221.      elsif (rx_output_strobe = '1') then
222.        rx_extended <= hold_extended;
223.        rx_released <= hold_released;
224.     --     rx_scan_code <= q(8 downto 1);
225.      -- elsif ctrl_key_on = '1' then
226.            --   rx_ascii <= ascii and x"1f";
227.      else
228.        rx_ascii <= ascii;
229.      end if;
230.     end if;
231.    end process;
232.    rx_output_proc : process( clk, reset, rx_shifting_done,
233.                        extended, released,
234.                                                    q,     ascii,
    rx_read )
235.    begin
236.     if (rx_shifting_done = '1') and (extended = '0') and (released = '0')
    then
```

```
237.       rx_output_event <= '1';
238.     else
239.       rx_output_event <= '0';
240.     end if;
241.
242.     if clk'event and clk = '0' then
243.       if reset = '1' then
244.             rx_output_strobe <= '0';
245.       elsif (rx_shifting_done = '1') and
246.           (rx_output_strobe = '0') and
247.                 (extended = '0') and
248.                                 (released = '0') and
249.                                 (hold_released = '0' ) and
250.           (ascii /= x"00" ) and
251.              ((TRAP_SHIFT_KEYS_PP = 0) or
252.               ( (q(8 downto 1) /= RIGHT_SHIFT) and
253.                       (q(8 downto 1) /= LEFT_SHIFT) and
254.                               (q(8 downto 1) /= CTRL_CODE) ) )then
255.         rx_output_strobe <= '1';
256.       elsif rx_read = '1' then
257.         rx_output_strobe <= '0';
258.       end if;
259.     end if;
260.     rx_data_ready <= rx_output_strobe;
261.   end process;
262.   shift_key_plus_code <= "000" & shift_key_on & "0" & q(7 downto
      1);
263.   shift_map : process( shift_key_plus_code)
264.   begin
265.
266.     if(asciimode='0') then
267.     case shift_key_plus_code is
268.     when x"066" => ascii <= x"11";  -- Backspace ("backspace" key)
269.     when x"166" => ascii <= x"11";  -- Backspace ("backspace" key)
270.     when x"076" =>                                  --esc
271.         ascii <= x"fe";
272.     when x"005" =>                      --f1
273.         ascii <=x"fd";
274.     when x"006" =>          --f2
275.         ascii <=x"fc";
276.     when x"045" => ascii <= x"10";  -- 0
277.     when x"016" => ascii <= x"01";  -- 1
278.     when x"01e" => ascii <= x"02";  -- 2
279.     when x"026" => ascii <= x"03";  -- 3
280.     when x"025" => ascii <= x"04";  -- 4
281.     when x"02e" => ascii <= x"05";  -- 5
282.     when x"036" => ascii <= x"06";  -- 6
```

```
283.      when x"03d" => ascii <= x"07";  -- 7
284.      when x"03e" => ascii <= x"08";  -- 8
285.      when x"046" => ascii <= x"09";  -- 9
286.      when x"01c" => ascii <= x"0a";  -- a
287.      when x"032" => ascii <= x"0b";  -- b
288.      when x"021" => ascii <= x"0c";  -- c
289.      when x"023" => ascii <= x"0d";  -- d
290.      when x"024" => ascii <= x"0e";  -- e
291.      when x"02b" => ascii <= x"0f";  -- f
292.      when others => ascii <= x"ff";  -- 0xff used for unlisted characters.
293.      end case;
294.     else
295.
296.      case shift_key_plus_code is
297.      when x"066" => ascii <= x"11";  -- Backspace ("backspace" key)
298.      when x"166" => ascii <= x"11";  -- Backspace ("backspace" key)
299.      when x"076" =>                                  --esc
300.          ascii <= x"fe";
301.      when x"006" =>          --f1        encrypt
302.          ascii <=x"fc";
303.      when x"029" => ascii <= x"20";  -- Space
304.      when x"129" => ascii <= x"20";  -- Space
305.       when others => ascii <= x"ff";   -- 0xff used for unlisted
    characters.
306.
307.       end case;
308.      end if;
309.     end process;
310.
311.     end my_ps2_keyboard;
```

F. AES_Core input output proses

```
1.  begin
2.  status_ext<=status;
3.
4.  output2LCD: process(clk)
5.    begin
6.     if(clk'event and clk='1') then
7.          case intLCD2Core is
8.            when 0 =>
9.                    Core2LCD<=INT_REGS(0,0);
10.                 when 1 =>
11.                   Core2LCD<=INT_REGS(1,0);
12.                 when 2 =>
13.                   Core2LCD<=INT_REGS(2,0);
14.                 when 3 =>
```

```
15.                 Core2LCD<=INT_REGS(3,0);
16.         when 4 =>
17.                 Core2LCD<=INT_REGS(0,1);
18.               when 5 =>
19.                 Core2LCD<=INT_REGS(1,1);
20.               when 6 =>
21.                 Core2LCD<=INT_REGS(2,1);
22.               when 7 =>
23.                 Core2LCD<=INT_REGS(3,1);
24.         when 8 =>
25.                 Core2LCD<=INT_REGS(0,2);
26.               when 9 =>
27.                 Core2LCD<=INT_REGS(1,2);
28.               when 10 =>
29.                 Core2LCD<=INT_REGS(2,2);
30.               when 11 =>
31.                 Core2LCD<=INT_REGS(3,2);
32.         when 12 =>
33.                 Core2LCD<=INT_REGS(0,3);
34.               when 13 =>
35.                 Core2LCD<=INT_REGS(1,3);
36.               when 14 =>
37.                 Core2LCD<=INT_REGS(2,3);
38.               when 15 =>
39.                 Core2LCD<=INT_REGS(3,3);
40.               when others=>
41.       end case;
42.    end if;
43. end process;
44.
45. asciimode<=int_asciimode;
46.
47. opcodes: process (opcode, clk, reset)
48.           variable inv0,inv1,inv2,inv3: std_logic_vector(10 downto 0);
49.                 variable tmp: std_logic_vector(3 downto 0);
50.           begin
51.              if(reset='0') then
52.                                       int_asciimode<='1';
53.                                       ps2inputpointer<= "0" & x"0";
54.                                       ps2cnt<=0;
55.                                       rx_read<='0';
56.                                       loaddata<='0';
57.                                       loadencrypt<='0';
58.                                       rambus<=(others=>'Z');
59.                                 --                                    INT_REGS<=
    ((x"00",x"44",x"88",x"cc"),
60.           --              (x"11",x"55",x"99",x"dd"),
```

```
61.          --              (x"22",x"66",x"aa",x"ee"),

62.                                  --
    (x"33",x"77",x"bb",x"ff"));
63.                                  INT_REGS<=     ((x"20",x"20",x"20",x"20"),
64.                      (x"20",x"20",x"20",x"20"),
65.                      (x"20",x"20",x"20",x"20"),

66.
    (x"20",x"20",x"20",x"20"));
67.
68.    elsif(clk'event and clk='1') then
69.       loaddata<='0';
70.       loadencrypt<='0';
71.     case opcode is -- NOP
72.        when "0000" =>
73.                         --       rambus<=(others=>'Z');
74.                                   rambus<=(others=>'Z');
75.          status<="000"; --idle
76.          RCON_counter<=0;
77.          MAIN_counter<=0;
78.                         if(inverse='1') then
79.                           WORD_EXTREGPTR<=40;
80.               else
81.                           WORD_EXTREGPTR<=0;
82.                         end if;
83. ------------------------------------------------------------------
84. --
85. --  Captures Input From Keyboard      When Idle
86. --
87. ------------------------------------------------------------------
88.          if(ps2cnt=7) then
89.            rx_read<='0';
90.                    ps2cnt<=0;
91.          else
92.                    ps2cnt<=ps2cnt+1;
93.          end if;
94.       if(rx_data_ready='1' and rx_released='1' and rx_read='0' ) then
95.                  leds<=  not rx_ascii;
96.           rx_read<='1';
97.                         if(inputaccept='1') then
98.          case rx_ascii is
99.                         when x"fe" =>
100.                                 int_asciimode<=not int_asciimode;
101.                                 ps2inputpointer<= "0" & x"0";
102.              when x"fc" =>
103.                                  loaddata<='1';
```

```
104.                         ps2inputpointer<= "0" & x"0";
105.                 when x"fd" =>
106.                         loadencrypt<='1';
107.                         ps2inputpointer<= "0" & x"0";
108.             when x"ff" =>
109.                         --ignore undecoded keys
110.                     when x"11" =>      --backspace (clear)
111.                         if(int_asciimode='0') then
112.                       if(ps2inputpointer(4)='0') then
113.                           tmp  :=  ps2inputpointer(3  downto  0)-
    "0001";
114.                                 INT_REGS(
115.                                   uVecToInt(tmp(1 downto 0)),
116.                                             uVecToInt(tmp(3
    downto 2)))(3 downto 0)<=x"f";
117.                                 ps2inputpointer<='1'              &
    (ps2inputpointer(3 downto 0) - "0001");
118.                                 else
119.
    INT_REGS(uVecToInt(ps2inputpointer(1                          downto
    0)),uVecToInt(ps2inputpointer(3 downto 2)))(7 downto 4)<=x"f";
120.                                 ps2inputpointer(4)<='0';
121.                                 end if;
122.                                else
123.                                 tmp:=       ps2inputpointer(3
    downto 0) - "0001";
124.                                 INT_REGS(uVecToInt(tmp(1 downto
    0)),uVecToInt(tmp(3 downto 2)))<=x"20";
125.                                 ps2inputpointer(3  downto  0)<=
        tmp;
126.                               end if;
127.             when others=>
128.                         if(int_asciimode='0') then
129.                                 if(ps2inputpointer(4)='0') then
130.
    INT_REGS(uVecToInt(ps2inputpointer(1                          downto
    0)),uVecToInt(ps2inputpointer(3  downto  2)))(7  downto  4)<=rx_ascii(3
    downto 0);
131.                                 ps2inputpointer(4)<='1';
132.                                else
133.
    INT_REGS(uVecToInt(ps2inputpointer(1                          downto
    0)),uVecToInt(ps2inputpointer(3  downto  2)))(3  downto  0)<=rx_ascii(3
    downto 0);
134.                                 ps2inputpointer<='0'              &
    (ps2inputpointer(3 downto 0) + "0001");
135.                                 end if;
```

```
136.                                    else
137.
    INT_REGS(uVecToInt(ps2inputpointer(1                    downto
    0)),uVecToInt(ps2inputpointer(3 downto 2)))<=rx_ascii;
138.                                        ps2inputpointer(3       downto
    0)<=ps2inputpointer(3 downto 0) + "0001";
139.                                    end if;
140.                 end case;
141.                 end if;
142.             end if;
```

G. LCD_Display_Interface

```
1.  Begin
2.    LCD_RW <= '0';
3.    SF_D   <= nibble;
4.    LCD_E  <= enable;
5.    LCD_RS <= regsel;
6.
7.    data_selector: process(istate, dstate, digit, keypress)
8.    begin
9.
10.     case istate is
11.       when istep_two | istep_four | istep_six =>
12.         byte <= X"30";
13.       when istep_eight =>
14.         byte <= X"20";
15.       when function_set =>
16.         byte <= X"28";
17.       when entry_mode =>
18.         byte <= X"06";
19.       when control_display =>
20.         byte <= X"0C";
21.       when clear_display =>
22.         byte <= X"01";
23.       when others =>
24.         byte <= (others => '0');
25.     end case;
26.
27.     if istate = init_done then
28.       case dstate is
29.         when set_start_address =>
30.           byte <= X"80"; -- first char of first line
31.         when S18_1 =>
32.                     byte <= X"C0";
33.                 when S19_2 =>
34.                     byte <= hexbox(uVecToInt(Core2LCD(7 downto 4)));
```

```
35.                 when S19_3 =>
36.                         byte <= Core2LCD;
37.                 when S20_1 =>
38.                         byte <= hexbox(uVecToInt(Core2LCD(3 downto 0)));
39.                 when S22 =>
40.                         byte <= X"80";
41.                 when S23_1 =>
42.                         byte <= X"C0";
43.                 when S24 =>
44.                         byte <= hexbox(uVecToInt(Core2LCD(7 downto 4)));
45.                 when S25 =>
46.                         byte <= hexbox(uVecToInt(Core2LCD(3 downto 0)));
47.                 when S26 =>
48.                        byte <= X"00";
49.                 when others =>
50.           byte <= (others => '0');
51.       end case;
52.     end if;
53.
54.   end process data_selector;
55.
56.
57.   nibble_select: process (selnibble, byte)
58.   begin
59.     case selnibble is
60.       when '0' => -- pass lower nibble
61.         nibble <= byte(3 downto 0);
62.       when '1' => -- pass upper nibble
63.         nibble <= byte(7 downto 4);
64.       when others => -- nothing to do
65.     end case;
66.   end process nibble_select;
67.
68.   init_sm:    process    (istate,    idone,    timer_15ms,    timer_4100us,
    timer_100us,
69.                 timer_40us, timer_1640us, txdone )
70.   begin
71.     -- default assignments
72.     next_istate     <= istate;
73.     next_idone      <= idone;
74.
75.     case istate is
76.
77.       when istep_one => -- wait here for 15 ms
78.
79.         if (timer_15ms = '1') then
80.           next_istate    <= istep_two;
```

```
81.       end if;
82.
83.     when istep_two => -- write nibble (0x3)
84.
85.       if (txdone = '1') then
86.         next_istate <= istep_three;
87.       end if;
88.
89.     when istep_three => -- wait here for 4100 us
90.
91.       if (timer_4100us = '1') then
92.         next_istate    <= istep_four;
93.       end if;
94.
95.     when istep_four => -- write nibble (0x3)
96.
97.       if (txdone = '1') then
98.         next_istate  <= istep_five;
99.       end if;
100.
101.        when istep_five => -- wait here for 100 us
102.
103.          if (timer_100us = '1') then
104.            next_istate   <= istep_six;
105.          end if;
106.
107.        when istep_six => -- write nibble (0x3)
108.
109.          if (txdone = '1') then
110.            next_istate <= istep_seven;
111.          end if;
112.
113.        when istep_seven => -- wait here for 40 us
114.
115.          if (timer_40us = '1') then
116.            next_istate   <= istep_eight;
117.          end if;
118.
119.        when istep_eight => -- write nibble (0x2)
120.
121.          if (txdone = '1') then
122.            next_istate  <= istep_nine;
123.          end if;
124.
125.        when istep_nine => -- wait here for 40 us
126.
127.          if (timer_40us = '1') then
```

```
128.            next_istate   <= function_set;
129.          end if;
130.
131.        when function_set => -- istep 10:
132.                             -- write data (0x28)
133.
134.         if (txdone = '1') then
135.           next_istate <= entry_mode;
136.         end if;
137.
138.        when entry_mode => -- istep 11
139.                           -- write data 0x06
140.
141.          if (txdone = '1') then
142.            next_istate <= control_display;
143.          end if;
144.
145.        when control_display => -- istep 12
146.                               -- enable display, disable cursor, disableblinking
147.                               -- write data 0x0C
148.
149.          if (txdone = '1') then
150.            next_istate <= clear_display;
151.          end if;
152.
153.        when clear_display => -- istep 13
154.                              -- write data 0x01
155.
156.          if (txdone = '1') then
157.            next_istate <= init_done; -- init. done
158.          end if;
159.
160.        when init_done =>
161.          if (timer_1640us = '1') then
162.            next_idone     <= '1';
163.          end if;
164.
165.        when others => -- nothing to do
166.
167.      end case;
168.
169.    end process init_sm;
170.
171.
172.    tx_m: process(istate, txcount, byte, selnibble, enable, txdone,
173.                  idone, dstate)
174.    begin
```

```
175.        next_selnibble <= selnibble;
176.        next_txdone    <= txdone;
177.        next_txcount   <= txcount;
178.        next_enable    <= enable;
179.
180.
181.        case istate is
182.          when istep_one | istep_three | istep_seven | istep_nine =>
183.            next_selnibble <= '1'; -- pass hign nibble
184.            -- transmit a nibble
185.          when istep_two | istep_four | istep_six | istep_eight =>
186.            next_txcount <= txcount + 1;
187.            if (txcount = 1) then
188.              next_enable <= '1';
189.            end if;
190.            if (txcount = 10) then
191.              next_enable <= '0';
192.              next_txdone <= '1';
193.            end if;
194.            if (txcount = 11) then
195.              next_txcount    <= 0;
196.              next_txdone     <= '0';
197.              --next we could pass zeros on the SF_D bus
198.            end if;
199.          -- transmit a byte
200.          when function_set | entry_mode | control_display |
201.               clear_display | init_done =>
202.
203.
204.            if (istate /= init_done or
205.               (istate = init_done and
206.               (dstate = set_start_address or
207.                                    dstate = S18 or
208.                                    dstate = S18_1 or
209.                                    dstate = S19 or
210.                                    dstate = S19_1 or
211.                                    dstate = S19_2 or
212.                                    dstate = S19_3 or
213.                                    dstate = S20 or
214.                                    dstate = S20_1 or
215.                                    dstate = S20_2 or
216.                                    dstate = S21 or
217.                                    dstate = S22 or
218.                                    dstate = S23 or
219.                                    dstate = S23_1 or
220.                                    dstate = S24 or
221.                                    dstate = S25 or
```

```
222.                                    dstate = S26
223.              ))) then
224.
225.              next_txcount <= txcount + 1;
226.              if (txcount = 1) then
227.                next_enable <= '1';
228.              end if;
229.              if (txcount = 10) then
230.                next_enable <= '0';
231.              end if;
232.              if (txcount = 11) then
233.                -- next we could pass zeros on the SF_D bus
234.              end if;
235.
236.              if (txcount = 58) then    -- 10 + 1 + 50 - 2 = 58
237.                next_selnibble <= '0'; -- pass lower nibble
238.              end if;
239.              if (txcount = 60) then
240.                next_enable <= '1';
241.              end if;
242.              if (txcount = 69) then
243.                next_enable <= '0';
244.              end if;
245.              if(txcount = 70) then -- done with the lower nibble data
246.                -- next we could pass zeros on the SF_D bus
247.              end if;
248.
249.              if (txcount = 2067) then
250.                next_txdone <= '1';
251.              end if;
252.              if (txcount = 2068) then -- 69 + 1 + 2000 - 2 =
253.                next_txcount   <= 0;
254.                next_txdone    <= '0';
255.                next_selnibble <= '1'; -- pass upper nibble
256.              end if;
257.            end if;
258.          when others => --nothing to do
259.        end case;
260.
261.      end process tx_m;
262.
263.
264.      display_sm: process(dstate, txdone, idone, regsel, txcount, lcdcnt,
     currentmode, inputaccept,keypress, asciimode,intLCD2Core)
265.      begin
266.
267.        -- by default hold state
```

```
268.        next_dstate <= dstate;
269.        next_regsel <= regsel;
270.              lcdcnt <= intLCD2Core;
271.              next_currentmode <= currentmode;
272.              next_inputaccept <= inputaccept;
273.
274.        if txcount = 11 then
275.          next_regsel <= '0';
276.        end if;
277.        if txcount = 58 then
278.          next_regsel <= idone; --high for active write dstates, low for
    istates
279.        end if;
280.        if txcount = 70 then
281.          next_regsel <= '0';
282.        end if;
283.
284.        case dstate is
285.
286.          when didle =>
287.            next_regsel <= '0'; -- must be low for active istates
288.            if (idone = '1') then
289.              next_dstate  <= set_start_address;
290.              next_regsel  <= '0'; -- must be low for address commands
291.            end if;
292.
293.          when set_start_address => -- start the text at the first
294.            next_regsel <= '0';                    -- location of the first line
295.            if (txdone = '1') then
296.                              lcdcnt <= 0;
297.                              next_dstate <= S18;
298.                    end if;
299.
300.                    when S18 =>
301.
302.                              if (intLCD2Core = 8 and asciimode = '0') then
303.                                        next_dstate <= S18_1;
304.                                        next_regsel <= '0';
305.                              else
306.                                        next_dstate <= S19;
307.                              end if;
308.
309.                    when S18_1 =>
310.                              next_regsel <= '0';
311.                              if (txdone = '1') then
312.                                        next_dstate <= S19;
313.                              end if;
```

```
314.
315.            when S19 =>
316.                    if (opcode = "1111") then
317.                            next_dstate <= S19_1;
318.
319.                    else
320.                            if (asciimode = '0') then
321.                                    next_dstate <= S19_2;
322.                                    next_regsel <= '1';
323.                            else
324.                                    next_dstate <= S19_3;
325.                                    next_regsel <= '1';
326.                            end if;
327.                    end if;
328.
329.            when S19_1 =>
330.
331.                            next_dstate <= S22;
332.                            next_regsel <= '0';
333.
334.
335.            when S19_2 =>
336.                    if (txdone = '1') then
337.                            next_dstate <= S20;
338.                    end if;
339.
340.            when S19_3 =>
341.                    if (txdone = '1') then
342.                            next_dstate <= S20;
343.                    end if;
344.
345.            when S20 =>
346.                    if (opcode ="1111") then
347.                            next_dstate <= S19_1;
348.                            next_regsel <= '1';
349.                    else
350.                            if (asciimode = '0') then
351.                                    next_dstate <= S20_1;
352.                                    next_regsel <= '1';
353.                            else
354.                                    next_dstate <= S20_2;
355.
356.                            end if;
357.                    end if;
358.
359.            when S20_1 =>
360.                    if (txdone = '1') then
```

```
361.                    next_dstate <= S21;
362.
363.            end if;
364.
365.        when S20_2 =>
366.
367.                    next_dstate <= S21;
368.
369.
370.        when S21 =>
371.
372.                    if(intLCD2Core =15 ) then
373.                        lcdcnt <= 0;
374.                      next_dstate <= set_start_address;
375.                        next_regsel <= '0';
376.                    else
377.                        lcdcnt <= intLCD2Core + 1;
378.                        next_dstate <= S18;
379.
380.                    end if;
381.                    if (opcode = "1111") then
382.                        next_dstate <= S22;
383.                        next_regsel <= '0';
384.                    end if;
385.
386.        when S22 =>
387.            next_regsel <= '0';
388.            if (txdone = '1') then
389.                lcd_cnt <= 0;
390.                next_dstate <= S23;
391.            end if;
392.
393.         when S23 =>
394.
395.            if (intLCD2Core = 8 ) then
396.                next_dstate <= S23_1;
397.                next_regsel <= '0';
398.            else
399.                next_dstate <= S24;
400.                next_regsel <= '1';
401.            end if;
402.
403.        when S23_1 =>
404.            next_regsel <= '0';
405.            if (txdone = '1') then
406.                next_dstate <= S24;
407.                next_regsel <= '1';
```

```
408.                        end if;
409.
410.                when S24 =>
411.                        if (txdone = '1') then
412.                                next_dstate <= S25;
413.                                next_regsel <= '1';
414.                        end if;
415.
416.                when S25 =>
417.                        if (txdone = '1') then
418.                                next_dstate <= S26;
419.                                next_regsel <= '0';
420.                        end if;
421.
422.                when S26 =>
423.                                if (intLCD2Core = 15) then
424.                                        lcdcnt <= 0;
425.                                  next_dstate <= S22;
426.                                        next_regsel <= '0';
                                    else
427.                                        lcdcnt <= intLCD2Core + 1;
428.                                        next_dstate <= S23;
429.                                end if;
430.       when others => -- nothing to do;
431.
432.      end case;
433.
434.    end process display_sm;
435.
436.
437.    registers: process(rst, clk, keypress)
438.    begin
439.      if rst = '0' then
440.        istate    <= istep_one;
441.        dstate    <= didle;
442.        idone     <= '0';
443.        count     <=  0;
444.        txcount   <=  0;
445.        selnibble <= '1';
446.        enable    <= '0';
447.        txdone    <= '0';
448.        regsel    <= '0';
449.        cnt       <= 0;
450.                    intLCD2Core <= 0;
451.                    inputaccept <= '1';
452.                    currentmode <= '1';
453.
```

```
454.        elsif clk = '1' and clk'event then
455.          istate    <= next_istate;
456.          dstate    <= next_dstate;
457.          idone     <= next_idone;
458.          count     <= next_count;
459.          txcount   <= next_txcount;
460.          selnibble <= next_selnibble;
461.          enable    <= next_enable;
462.          txdone    <= next_txdone;
463.          regsel    <= next_regsel;
464.          digit     <= next_digit;
465.          cnt       <= next_cnt;
466.                       inputaccept <= next_inputaccept;
467.                       currentmode <= next_currentmode;
468.                       intLCD2Core <= lcdcnt;
469.      end if;
470.       end process registers;
471.     end rtl;
```