ABSTRAK

ARI KUSYANTI, Jurusan Teknik Elektro, Fakultas Teknik, Univeritas Brawijaya, April 2007, *Desain dan Implementasi Stream Cipher Cryptography Processor Algoritma Grain*, Dosen Pembimbing : Ir. Heru Nurwarsito, M.Kom dan Ir. Primantara HT.

Perkembangan teknologi komunikasi dewasa ini berkembang dengan sangat cepat dengan adanya teknologi internet. Hal ini melibatkan peralatan komunikasi mulai dari peralatan dengan kabel (*wired*) sampai dengan peralatan nirkabel (*wireless*) yang terhubung secara global. Dengan terhubungnya peralatan komunikasi dalam satu jaringan, akan membuka peluang bagi pihak yang tidak bertanggung jawab untuk melakukan pengambilan, penambahan, penghapusan ataupun penggantian data. Untuk mencegah hal tersebut, diperlukan sebuah metode untuk melindungi data yang disebut dengan *cryptography*.

Salah satu algoritma cryptography yang digunakan adalah Grain. Grain merupakan *stream cipher* yang dibuat oleh Martin Hell, Thomas Johansson, dan Will Meier dari Lund University, Swedia, untuk diikutsertakan dalam proposal eSTREAM sebuah proyek *security* Eropa. Grain diharapkan akan menggantikan A5/1 pada GSM *System*.

Algoritma ini diimplementasikan dalam bentuk *hardware* dengan menggunakan bahasa pemrograman VHDL, karena kecepatan eksekusi pada *hardware* lebih cepat daripada *software*. S*tream cipher cryptography processor* algoritma Grain ini terdiri atas 12 pin utama, 10 pin sebagai pin masukan dan 2 pin keluaran, serta pin untuk catu daya dan *ground*. S*tream cipher cryptography processor* algoritma Grain ini memiliki 3 pilihan algoritma, yaitu Grain v0, Grain v1 dan Grain 128. Implementasi s*tream cipher cryptography processor* algoritma Grain ini menghabiskan 5.193 *logic gates*, dengan *throughput* 0,14 Gbps dan frekuensi maksimum 138,735 MHz pada *device* Xilinx *Spartan3E XC3S500 E* dengan *package FG320*.

Keystream yang dihasilkan s*tream cipher cryptography processor* algoritma Grain ini telah teruji kebenarannya dengan menggunakan *test vector* Grain dan juga telah memenuhi pengujian tingkat keacakan bit berdasarkan *National Institute of Standard and Technology* [NIST] melalui pengujian *Frequency Test, Frequency Test within a Block, Runs Test, The Longest Run in a Block* dan *Binary Matrix Test* dengan nilai parameter *P-value* untuk masing-masing test telah memenuhi standar keacakan yaitu lebih dari atau sama dengan 0,01.

S*tream cipher cryptography processor* algoritma Grain ini telah siap digunakan untuk aplikasi lain yang membutuhkan keamanan data, dan dapat digunakan sebagai dasar penelitian lebih lanjut untuk digabungkan dengan komponen lain seperti: *Multi-Purpose Processor, Memory* dan *I/O Devices* untuk membentuk sistem yang lengkap yang sekarang berkembang dengan pesat yang disebut dengan *System-on-Chip* (SoC).

Kata kunci:
***cryptography, stream cipher,* algoritma Grain, VHDL, *cryptography processor*.**

DESAIN DAN IMPLEMENTASI
STREAM CIPHER CRYPTOGRAPHY PROCESSOR
ALGORITMA GRAIN

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan

Memperoleh gelar Sarjana Teknik

Disusun oleh :

ARI KUSYANTI

NIM. 0210630020

DEPARTEMEN PENDIDIKAN NASIONAL

UNIVERSITAS BRAWIJAYA

FAKULTAS TEKNIK

MALANG

2007

DAFTAR GAMBAR

DAFTAR ISI

DAFTAR LAMPIRAN

DAFTAR TABEL

DESAIN DAN IMPLEMENTASI
STREAM CIPHER CRYPTOGRAPHY PROCESSOR
ALGORITMA GRAIN

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan
Memperoleh gelar Sarjana Teknik

Disusun oleh:

ARI KUSYANTI

NIM. 0210630020

DOSEN PEMBIMBING

Ir. Heru Nurwarsito, M.Kom	Ir. Primantara Hari Trisnawan
NIP. 131 879 033	NIP. 132 090 390

DESAIN DAN IMPLEMENTASI

STREAM CIPHER CRYPTOGRAPHY PROCESSOR

ALGORITMA GRAIN

Disusun oleh:

ARI KUSYANTI

NIM. 0210630020

Skripsi ini telah diuji dan dinyatakan lulus

pada tanggal 4 Mei 2007

DOSEN PENGUJI

Ir. Muhammad Aswin
NIP. 131 879 045

Herman Tolle, ST., MT.
NIP. 132 283 206

Arief Andy Soebroto, ST., M.Kom
NIP. 132 231 567

Ir. Bambang Siswojo
NIP. 132 149 320

Mengetahui,

Ketua Jurusan Teknik Elektro

Ir. Purwanto, MT.
NIP. 131 574 847

PENGANTAR

Alhamdulillah, puji syukur kehadirat Allah SWT atas rahmat dan hidayah-Nya sehingga penulis dapat menyelesaikan tugas akhir dengan judul "**Desain dan Implementasi *Stream Cipher Cryptography Processor* Algoritma Grain**".

Tidak banyak yang bisa penulis sampaikan kecuali ungkapan terima kasih kepada berbagai pihak yang telah dengan tulus ikhlas memberikan bimbingan, arahan, dan dukungan hingga penulisan tugas akhir ini dapat terselesaikan. Terima kasih kepada Bapak Ir. Purwanto, MT selaku Ketua Jurusan Teknik Elektro.

Terima kasih kepada Bapak Ir. Heru Nurwarsito, M.Kom dan Bapak Ir. Primantara HT selaku pembimbing Tugas Akhir ini. Terima kasih kepada Bapak Dr. Marjono, M. Phil. selaku dosen referensi.

Kepada Ayahanda Asmari dan Ibunda Mariami serta kakak Popong Effendrik. Terima kasih atas segala do'a, dukungan, semangat dan kasih sayang yang diberikan.

Terima kasih kepada GE Foundation yang telah memberikan beasiswa selama masa perkuliahan, dan memberikan bantuan dana untuk penyelesaian Tugas Akhir ini. Ibu Dr. Irid Agoes selaku *director* IIEF dan GE Foundation. Ibu Fenty Agtiffantono, Bapak Wahono Kolopaking dan Ibu Eldyna Wardhana selaku *program director* GE Foundation. Terima kasih atas dukungan dan bantuannya selama ini.

Kepada teman-teman yang selalu mendukung dan memberi semangat serta do'a, Rizka Zakiyah, Indriati, Vivi Sesma P. Terima kasih untuk Nadia, Rida, Prima, Dian, Ariza, Ina, Anton, Erik, Said atas dukungan dan do'anya. Teman-teman di Teknik Elektro, angkatan 2002 khususnya.

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

Malang, April 2007

Penulis

Allah, tidak ada Tuhan (yang berhak disembah)
melainkan Dia Yang Hidup kekal lagi terus menerus mengurus (mahkluk-Nya);
tidak mengantuk dan tidak tidur.
Kepunyaan-Nya apa yang ada di langit dan di bumi.
Tiada yang dapat memberi syafa'at di sisi Allah tanpa izin-Nya.
Allah mengetahui apa-apa yang di hadapan mereka dan di belakang mereka,
dan mereka tidak mengetahui apa-apa dari ilmu Allah
melainkan apa yang dikehendaki-Nya.
Kursi Allah meliputi langit dan bumi.
Dan Allah tidak merasa berat memelihara keduanya,
dan Allah Maha Tinggi lagi Maha Besar.
(QS. 2:255)

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

DAFTAR LAMPIRAN

DAFTAR ISTILAH

Authentification
Suatu pelayanan yang digunakan untuk mengidentifikasi entitas pesan yang dikirim secara keseluruhan, termasuk asal pengirim pesan.

Block Cipher
Algoritma kunci simetrik yang mengenkripsi setiap blok dalam *plaintext* menjadi *ciphertext* dalam satu waktu.

Ciphertext
Data terenkripsi yang tidak dapat dibaca.

Confidentiality
Suatu pelayanan yang diberikan untuk menjaga kerahasiaan isi dari suatu pesan hanya untuk orang yang berhak.

Cryptography
Sebuah metode yang digunakan untuk mengamankan dan melindungi data dari pihak yang tidak berhak.

Data Integrity
Suatu pelayanan yang diberikan untuk mencegah pihak yang tidak berhak untuk mengubah isi data.

Dekripsi
Proses untuk mengubah *ciphertext* menjadi *plaintext*.

Enkripsi
Proses untuk mengubah *plaintext* menjadi *ciphertext*.

Initialization Vector
Sequence bit yang digunakan untuk menginisialisasi suatu *stream cipher*.

Irreducible Polynomial
Polinomial yang tidak dapat difaktorkan menjadi polinomial dengan derajat yang lebih rendah.

Keystream
Kunci yang berupa *binary sequence* yang dihasilkan *stream cipher*.

Linear Feedback Shift Register
Sebuah *shift register* yang menggunakan fungsi *linear* sebagai nilai kembaliannya.

Non Linear Feedback Shift Register
Sebuah *shift register* yang menggunakan fungsi *non linear* sebagai nilai kembaliannya.

Non-Repudiation

Suatu pelayanan yang diberikan untuk mencegah sebuah entitas dari pengingkaran komitmen dan aksi sebelumnya.

Plaintext

Data asli yang akan dienkripsi.

P-value

Probabilitas yang dipilih sebagai nilai acuan untuk menentukan tingkat keacakan dalam menetukan tingkat keacakan bit.

Symmetric Key

Algoritma yang menggunakan satu kunci yang sama untuk melakukan enkripsi dan dekripsi.

Stream Cipher

Algoritma kunci simetrik yang mengenkripsi setiap bit dalam *plaintext* menjadi *ciphertext* dalam satu waktu.

VHDL

Very High Speed Integrated Circuit (VHSIC) Hardware Description Language. Bahasa pemrograman untuk mendeskripsikan fungsi, perilaku masukan dan keluaran suatu rancangan rangkaian digital

ABSTRAK

ARI KUSYANTI, Jurusan Teknik Elektro, Fakultas Teknik, Univeritas Brawijaya, April 2007, *Desain dan Implementasi Stream Cipher Cryptography Processor Algoritma Grain*, Dosen Pembimbing : Ir. Heru Nurwarsito, M.Kom dan Ir. Primantara HT.

Perkembangan teknologi komunikasi dewasa ini berkembang dengan sangat cepat dengan adanya teknologi internet. Hal ini melibatkan peralatan komunikasi mulai dari peralatan dengan kabel (*wired*) sampai dengan peralatan nirkabel (*wireless*) yang terhubung secara global. Dengan terhubungnya peralatan komunikasi dalam satu jaringan, akan membuka peluang bagi pihak yang tidak bertanggung jawab untuk melakukan pengambilan, penambahan, penghapusan ataupun penggantian data. Untuk mencegah hal tersebut, diperlukan sebuah metode untuk melindungi data yang disebut dengan *cryptography*.

Salah satu algoritma cryptography yang digunakan adalah Grain. Grain merupakan *stream cipher* yang dibuat oleh Martin Hell, Thomas Johansson, dan Will Meier dari Lund University, Swedia, untuk diikutsertakan dalam proposal eSTREAM sebuah proyek *security* Eropa. Grain diharapkan akan menggantikan A5/1 pada GSM *System*.

Algoritma ini diimplementasikan dalam bentuk *hardware* dengan menggunakan bahasa pemrograman VHDL, karena kecepatan eksekusi pada *hardware* lebih cepat daripada *software*. S*tream cipher cryptography processor* algoritma Grain ini terdiri atas 12 pin utama, 10 pin sebagai pin masukan dan 2 pin keluaran, serta pin untuk catu daya dan *ground*. S*tream cipher cryptography processor* algoritma Grain ini memiliki 3 pilihan algoritma, yaitu Grain v0, Grain v1 dan Grain 128. Implementasi s*tream cipher cryptography processor* algoritma Grain ini menghabiskan 5.193 *logic gates*, dengan *throughput* 0,14 Gbps dan frekuensi maksimum 138,735 MHz pada *device* Xilinx *Spartan3E XC3S500 E* dengan *package FG320*.

Keystream yang dihasilkan s*tream cipher cryptography processor* algoritma Grain ini telah teruji kebenarannya dengan menggunakan *test vector* Grain dan juga telah memenuhi pengujian tingkat keacakan bit berdasarkan *National Institute of Standard and Technology* [NIST] melalui pengujian *Frequency Test, Frequency Test within a Block, Runs Test, The Longest Run in a Block* dan *Binary Matrix Test* dengan nilai parameter *P-value* untuk masing-masing test telah memenuhi standar keacakan yaitu lebih dari atau sama dengan 0,01.

S*tream cipher cryptography processor* algoritma Grain ini telah siap digunakan untuk aplikasi lain yang membutuhkan keamanan data, dan dapat digunakan sebagai dasar penelitian lebih lanjut untuk digabungkan dengan komponen lain seperti: *Multi-Purpose Processor, Memory* dan *I/O Devices* untuk membentuk sistem yang lengkap yang sekarang berkembang dengan pesat yang disebut dengan *System-on-Chip* (SoC).

Kata kunci:
***cryptography, stream cipher,* algoritma Grain, VHDL, *cryptography processor*.**

REFERENSI

[Ash-90] Ashenden, P.J. 1990. *The VHDL Cookbook*. First Edition. Dept. Computer Science, University of Adelaide, South Australia.

[Bir-05] Biryukov, A. 2005. *Block Cipher and Stream Cipher : The State of the Art.* Katholieke Universiteit Leuven, Dept. ESAT/SCD-COSIC, Belgium.

[Ecr- 05] ECRYPT Network of Excellence eSTREAM – Update 1. September 2005.

[EJ-02] Ekdahl, P., Johansson T. 2002. *Another Attack on A5/1*. IEEE Transactions on Information Theory.

[Gar-05] Garret, Paul. 2005. *Abstract Algebra : Lectures and Worked Examples for Graduate Course*. University of Minnesota.

[GL-05] Gurkaynak, F. K. and Luethi, P. 2005. *Recommendations for Hardware Evaluation of Cryptographic Algorithms.* Integrated System Laboratory ETH Zurich, Switzerland.

[Han-00] Hankerson, D.R. et al. 2000. *Coding Theory and Cryptography, The Essentials*. Marcell Dekker, Inc

[HJM-05] Hell, M. Johansson, T. and Meier, W. 2005. *Grain – Stream Cipher for Constrained Environments*. Dept. of Information and Technology, Lund University, Sweden.

[HJM-06] Hell, M. Johansson, T. Maximov, A. and Meier, W. 2006. *A Stream Cipher Proposal: Grain 128*. Dept. of Information and Technology, Lund University, Sweden.

[Kea-02] Kean, Tom. 2002. *Cryptographic Rights Management of FPGA Intellectual Property Cores*. Algotronix Ltd. Edinburgh EH8 8YB United Kingdom. Security *FPGA 2002*.

[KLP-04] Kumar, S. Lemke, K. and Paar, C. 2004. *Some Thoughts about Implementation Properties of Stream Ciphers.* Horst Gortz Institute for IT Security, Ruhr Universitat Bochum, Germany.

[MOV-97] Menezes, Oorschot, v. P. and Vanstone, S. 1997. *Handbook of Applied Cryptography*. Boca Raton, CRC Press.

[NIS-04] National Institute of Standard and Technology (NIST). 2004. *Common Criteria for Information Technology Security Evaluation* (version 2.2) Revision 256.

[NIS-01] National Institute of Standard and Technology (NIST). 2001. *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications*. NIST Special Publication 800-22 (with revision dated May 15, 2001).

[Nyb-04] Nyberg, Kaysa. 2004. *Cryptographic Algorithms for UMTS*. Nokia Research Centre. European Congress on Computational Methods in Applied Sciences and Engineering 2004.

[Osw-05] Oswald, E. et al. 2005. *State of the Art in Hardware Architectur*e. ECRYPT European Network of Excellence in Cryptology, Information Socienty Technologies.

[Per-94] Perry, D.L. 1994. *VHDL*. McGraw Hill Book Co. International Edition.

[Ros-05] Rosen, KH. 2005. *Number Theory and Its Application*. AT&T Laboratories. Pearon Addition Wesley 2005.

[RRS-03] Ravi, S., Anand Raghunathan and Srimat Chakradhar. 2003. *Embedding Security in Wireless Embedded Systems*. IEEE Proceedings of the 16th International Conference on VLSI Design (VLSI'03)

[Rue-92] Rueppel, R. A. 1992. *Stream Cipher*. In G. J. Simmons editors. Contemporary Cryptology: The Science of Information Integrity. IEEE Press 1992.

[Rus-96] Rushanan, J. J. 1996. *A Tutorial on Finite Field and Binary m-Sequences*. The Mitre Corporation.

[Tur-05] Turan, M. S. 2005. *Statistical Analysis of Synchronous Stream Cipher*. Institute of Applied Mathematic, Middle East Technical University, Turkey.

[WP-04] Wollinger, Thomas and Christof Paar. 2004. *How Secure Are FPGAs in Cryptographic Applications?*. COSY Horst Gortz Institute for IT Security. Ruhr-Universitat Bochum, Germany.

[VAC-99] Visible Analyze Co. 1999. *Visible Analyze Tutorial*. Visible Analyze Corporation.

[Xil-06a] Xilinx Inc. 2006. *Programmable Logic Design. Quick Start Handbook*. Xilinx Inc.

[Xil-06b] Xilinx Inc. 2006. *Synthesis and Verification Design Guide*. Xilinx Inc.

[Xil-06c] Xilinx Inc. 2006. *Spartan-3E FPGA Family: Complete Data Sheet*. Xilinx Inc. DS312.

[Zen-04] Zenner, E. *Stream Cipher Criteria*. CRYPTICO A/S. 2004.

BAB I
PENDAHULUAN

1.1 Latar Belakang

Perkembangan teknologi komunikasi dewasa ini berkembang dengan sangat cepat dengan adanya teknologi internet. Hal ini melibatkan peralatan komunikasi mulai dari peralatan dengan kabel (*wired*) sampai dengan peralatan nirkabel (*wireless*) yang terhubung secara global.

Terhubungnya peralatan tersebut dalam satu jaringan secara global, didapatkan keuntungan dalam hal kecepatan dan biaya perpindahan data dari satu tempat di ujung dunia yang satu ke tempat di ujung dunia yang lain dengan sangat cepat dan murah.

Keuntungan ini harus dibayar mahal dengan kelemahan dalam hal keamanan data. Terhubungnya peralatan komunikasi dalam satu jaringan, akan membuka peluang bagi pihak yang tidak bertanggung jawab untuk melakukan pengambilan, penambahan, penghapusan ataupun penggantian data. Untuk mencegah hal tersebut, diperlukan sebuah metode untuk melindungi data yang disebut dengan *cryptography*. Dalam Gambar 1.1. dapat dilihat sebuah skema komunikasi secara sederhana. Dalam skema tersebut, *Alice* dan *Bob* adalah pihak yang melakukan komunikasi, sedangkan pihak yang merugikan adalah *Eve*. Dalam berkomunikasi, *Alice* dan *Bob* menggunakan saluran komunikasi umum yang juga digunakan oleh orang lain untuk berkomunikasi. Apabila proses komunikasi tersebut tidak dilindungi, maka orang yang tidak bertanggung jawab seperti *Eve* akan dapat sangat membahayakan keamanan data yang dikirimkan.

Tujuan *cryptography*, yaitu untuk memberikan pelayanan dalam hal *confidentiality* (kerahasiaan), *data integrity* (integritas data), *entity authentification* (autentifikasi entitas), dan *data origin authentication* (autentifikasi data) [MOV-97].

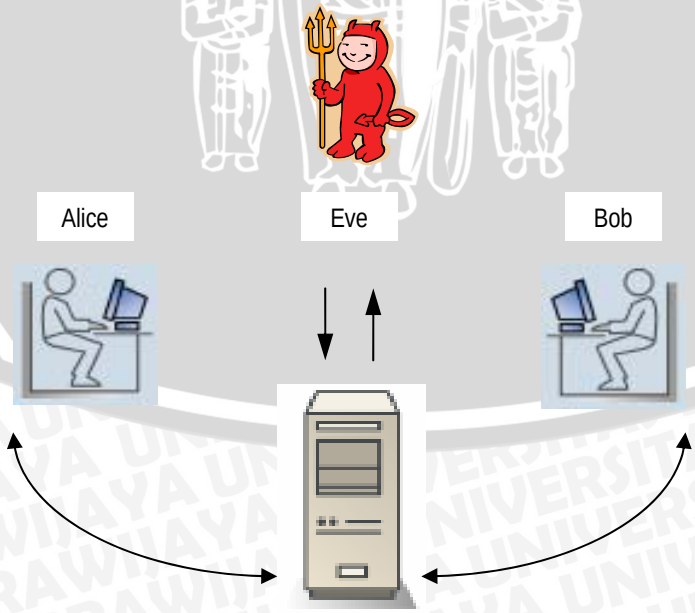
Cryptography mempunyai berbagai macam algoritma. Pemilihan algoritma *cryptography* yang sesuai sangat berpengaruh terhadap keandalan sistem. Selain algoritma yang digunakan, pengimplementasian algoritma tersebut pada *hardware* ataupun *software* akan berpengaruh terhadap kecepatan eksekusi algoritma. Kecepatan eksekusi algoritma pada *software* lebih rendah dibandingkan dengan kecepatan eksekusi pada *hardware* [Osw-05].

Pada *GSM System*, algoritma *cryptography* yang digunakan saat ini adalah A3, A8 dan A5. Algoritma A3 dan A8 digunakan untuk proses authentikasi pelanggan. Sedangkan, untuk proses *ciphering call* antara telepon dan *base station* digunakan algoritma A5 yang merupakan algoritma *stream cipher* [Nyb-04].

Tetapi pada tahun 2000, algoritma A5 ini telah berhasil dipecahkan oleh Biham dan Dunkelman [EJ-02]. Alex Biryukov, Adi Shamir dan David Wagner juga telah berhasil memecahkan A5/1 dalam waktu kurang dari satu detik dengan menggunakan PC 128 MB RAM [EJ-02].

Algoritma *cryptography* yang digunakan dalam tugas akhir ini adalah Grain. Mengingat Grain merupakan algoritma yang paling baru yang didesain pada tahun 2005 dan diharapkan akan menggantikan A5/1 pada GSM *System* yang telah dipecahkan. Grain merupakan kandidat terkuat dalam eSTREAM, sebuah proyek *security* di Eropa yang didirikan untuk mengembangkan *stream cipher*. Grain adalah algoritma *stream cipher* yang didesain oleh Martin Hell, Thomas Johansson, dan Will Meier dari Lund University, Swedia.

Dalam tugas akhir ini akan dilakukan desain dan implementasi *stream cipher cryptography processor* algoritma Grain menggunakan VHDL (*Very High Speed Integrated Circuit (VHSIC) Hardware Description Language*). Algoritma ini akan diimplementasikan pada *hardware*, yaitu IC (*Integrated Circuit*) menggunakan simulasi dengan program VHDL, memanfaatkan Xilinx *ISE WebPACK 8.1i*.



Gambar 1.1 Skema Komunikasi

1.2 Rumusan Masalah

Mengacu pada permasalahan yang telah diuraikan pada latar belakang, maka rumusan masalah dapat disusun sebagai berikut:

1. Pendesainan dan pengimplementasian *stream cipher cryptography processor* algoritma Grain dengan menggunakan VHDL (*Very High Speed Integrated Circuit (VHSIC) Hardware Description Language*) dengan memanfaatkan Xilinx *ISE WebPACK 8.1i*.
2. Pengujian hasil implementasi *stream cipher cryptography processor* algoritma Grain dengan menggunakan *ISE Simulator* yang terdapat pada Xilinx *ISE WebPACK 8.1i*.

1.3 Batasan Masalah

Dalam desain dan implementasi *stream cipher cryptography processor* algoritma Grain menggunakan VHDL ini dibatasi permasalahan yang mencakup:

1. Pembahasan algoritma *cryptography* secara terperinci hanya dilakukan pada *stream cipher* algoritma Grain.
2. Desain dan implementasi *stream cipher cryptography processor* algoritma Grain dengan menggunakan VHDL (*Very High Speed Integrated Circuit (VHSIC) Hardware Description Language)* dengan memanfaatkan Xilinx *ISE WebPACK 8.1i*.

1.4 Tujuan

Tujuan akhir dari penulisan tugas akhir ini adalah untuk mendesain dan mengimplementasikan *stream cipher cryptography processor* algoritma Grain menggunakan bahasa pemrograman VHDL.

1.5 Manfaat

Manfaat dari pengimplementasian *stream cipher cryptography processor* algoritma Grain ini adalah untuk menghasilkan *cryptography processor* yang bersifat *modular* sehingga dapat dimanfaatkan untuk aplikasi yang lebih kompleks.

1.6 Sistematika Pembahasan

Sistematika penulisan dalam tugas akhir ini sebagai berikut:

BAB I Pendahuluan

Memuat latar belakang, rumusan masalah, batasan masalah, tujuan, manfaat, dan sistematika pembahasan.

BAB II Dasar Teori

Membahas mengenai *cryptography* yaitu keamanan data, fungsi *cryptography*, skema enkripsi dan dekripsi dalam *secret key stream chipher, abstract algebra*, *randomness testing* dan algoritma Grain.

BAB III Metode Penelitian

Memuat sistematika pembahasan desain dan implementasi *stream cipher cryptography processor* algoritma Grain.

BAB IV Perancangan

Membahas tentang skema dan desain algoritma *stream cipher* Grain yang digunakan serta metode-metode dalam pengimplementasiannya.

BAB V Implementasi

Membahas pengimplementasian algoritma Grain dengan menggunakan bahasa perograman VHDL, dan melalui tahapan dalam perancangan IC, yaitu tahap spesifikasi, sintesis, simulasi, implementasi dan analisis serta tahap *place and route*.

BAB VI Pengujian dan Analisis

Memuat hasil pengujian terhadap hasil yang telah direalisasikan dengan menggunakan pengujian *keystrem* dengan menggunakan *test vector* Grain, dan *randomness testing* dari *National Institute of Standard and Technlogy* (NIST).

BAB VII Penutup

Memuat kesimpulan dan saran-saran.

BAB II
DASAR TEORI

2.1 *Cryptography*

Cryptography adalah sebuah studi matematis yang berhubungan dengan aspek keamanan informasi yaitu *confidentiality* (kerahasiaan), *data integrity* (integritas data), *entity authentification* (autentifikasi entitas), dan *data origin authentication* (autentifikasi data) [MOV-97].

Dari definisi *cryptography* tersebut dapat dijelaskan lebih lanjut tujuan dari *cryptography*, yaitu sebagai berikut:

1. *Confidentiality* (Kerahasiaan).

   Suatu pelayanan yang diberikan untuk menjaga isi dari suatu pesan hanya untuk orang yang berhak. Kerahasiaan adalah nama lain dari *confidentiality* atau privasi. Ada banyak pendekatan untuk menjamin *confidentiality*, mulai dari perlindungan secara fisik, sampai algoritma matematis yang membuat data tidak dapat dipahami.

2. *Data Integrity* (Integritas Data)

   Suatu pelayanan yang diberikan untuk mencegah pihak yang tidak berhak untuk mengubah isi data. Untuk memastikan integritas data, harus dapat dipastikan bahwa data tidak dimanipulasi oleh pihak yang tidak berhak. Manipulasi data tersebut dapat berupa penambahan, penghapusan ataupun penggantian data.

3. *Authentification* (Autentifikasi)

   Aspek *cryptography* dalam hal *authentification* terbagi menjadi dua yaitu *entity authentification* dan *data origin authentification*. *Entity authentification* digunakan untuk mengidentifikasi entitas pesan yang dikirim secara keseluruhan yang di dalamnya termasuk keaslian data, isi data, dan waktu mengirim. Sedangkan *data origin authentification* digunakan untuk mengidentifikasi asal pengirim pesan.

4. *Non-Repudiation*

   Suatu pelayanan yang diberikan untuk mencegah sebuah entitas dari pengingkaran komitmen dan aksi sebelumnya. Misalnya, sebuah entitas telah

memberikan otoritas pembayaran sebuah properti oleh entitas lain. Tetapi, kemudian mengingkari otoritas yang telah diberikan. Oleh karena itu dibutuhkan pihak lain yang dapat mencegah terjadinya hal tersebut.

Dalam era *cryptography* kuno, untuk menyembunyikan sebuah pesan, yang dirahasiakan adalah algoritma pesan tersebut. Salah satu contohnya adalah *Caesar Cipher,* yang digunakan dalam perang Romawi. Untuk menyampaikan pesan rahasia agar tidak diketahui musuh, algoritma *Caesar Cipher* inilah yang dirahasiakan.

Dewasa ini, *cryptography* dalam penggunaannya tidak lagi merahasiakan algoritma pesan, tetapi yang dirahasiakan adalah kunci yang digunakan. Algoritma pesan yang digunakan dipublikasikan, sehingga yang berperan dalam proses enkripsi dan dekripsi adalah kunci yang digunakan. Salah satu contohnya adalah Grain *Stream Cipher*, meskipun algoritma *cryptography* ini dipublikasikan, tetapi pesan masih dapat dijamin kerahasiaannya, karena untuk menyembunyikan pesan yang berperan adalah kunci yang digunakan. Metode penyembunyian kunci dan pempublikasian algoritma ini mempunyai tujuan agar semua lapisan masyarakat dapat menggunakan algoritma *cryptography* untuk keperluan mereka masing-masing dengan tingkat kemanan data yang sangat tinggi.

2.1.1 *Taxonomy Cryptography Primitives*

Dengan semakin kompleksnya kebutuhan *security* data, maka semakin banyak pula algoritma *cryptography* yang diperlukan untuk memenuhi kebutuhan tersebut. Secara garis besar, pembagian *primitives cryptography* dapat dilihat pada Gambar 2.1.

2.1.2 Proses Enkripsi dan Dekripsi

Dalam *cryptography*, terdapat dua proses utama yang digunakan untuk mengolah data asli (*plaintext)* menjadi data terenkripsi yang tidak dapat dibaca (*ciphertext)*, dan sebaliknya. Kedua proses tersebut adalah proses enkripsi dan dekripsi.

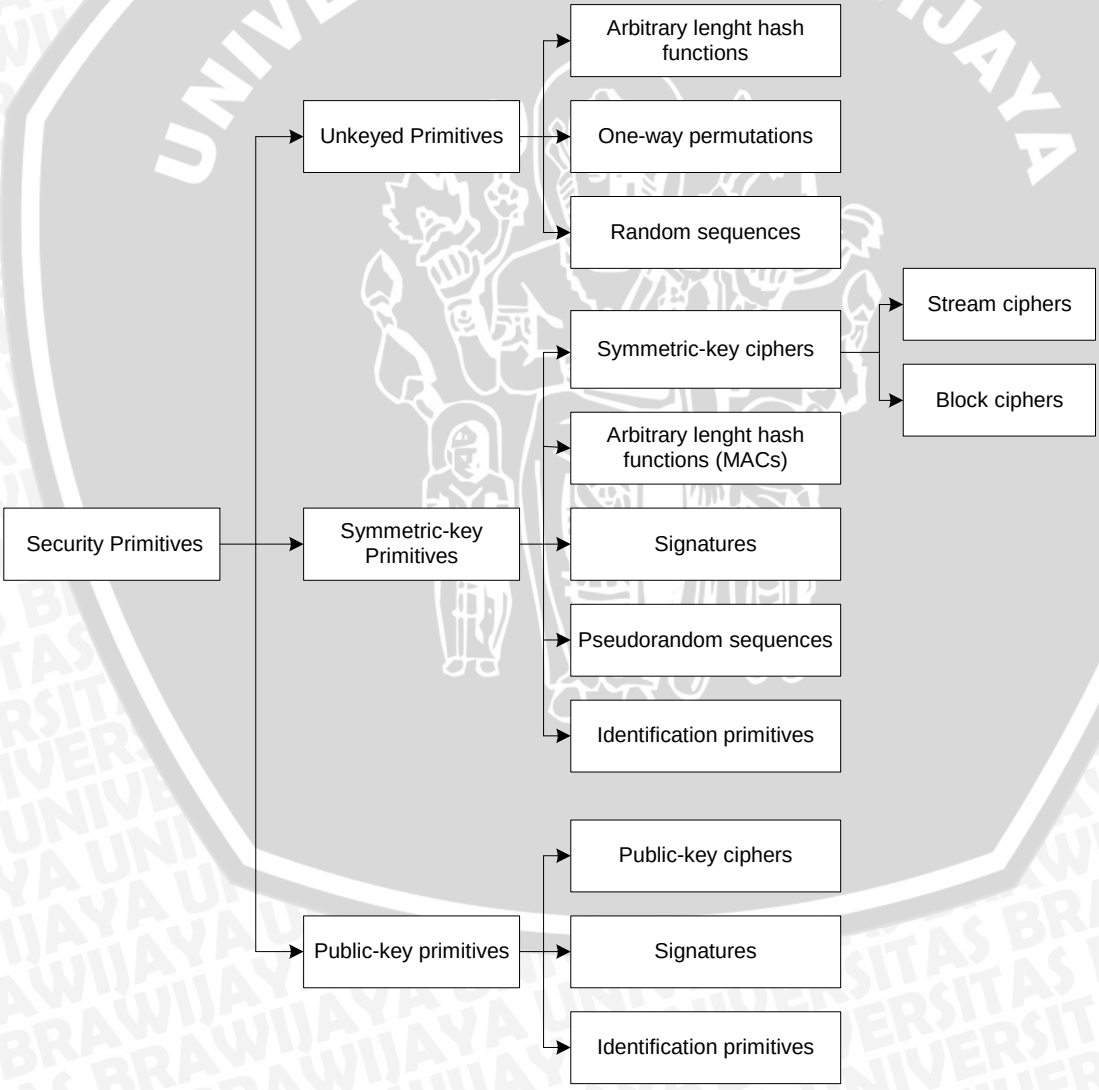
Enkripsi adalah sebuah proses untuk mengubah *plaintext* menjadi *ciphertext*. Proses kebalikan dari enkripsi yaitu dekripsi. Dekripsi adalah suatu proses untuk mengubah *ciphertext* menjadi *plaintext*.

Proses enkripsi dan dekripsi dapat dituliskan sebagai suatu fungsi matematis. Bila *plaintext* dinotasikan M, *ciphertext* dinotasikan C, maka untuk memperoleh *ciphertext,* dalam bentuk matematis proses enkripsi dapat ditulis [MOV-97]:

$$E(M) = C \tag{2.1}$$

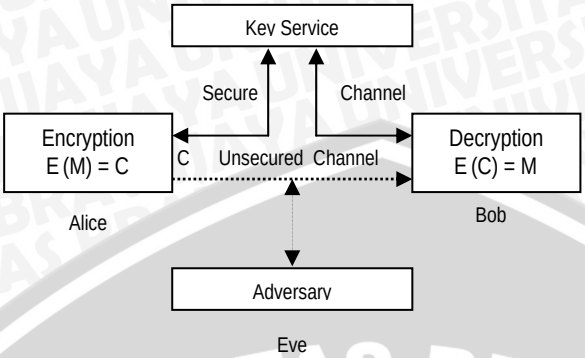
Untuk mendapatkan data asli, dilakukan proses dekripsi, dalam bentuk matematis dapat ditulis [MOV-97]:

$$D(C) = M \tag{2.2}$$



Gambar 2. 1 *Cryptography Primitives Taxonomy* [MOV-97]

Secara lebih jelas, dalam Gambar 2.2 dapat dilihat proses enkripsi dan proses dekripsi dalam suatu proses komunikasi.



Gambar 2.2 Proses Enkripsi dan Dekripsi [Han-00]

Salah satu bagian dalam *primitives cryptography* adalah algoritma kunci simetrik (*Simetric-key Primitives*) sebagaimana ditunjukkan dalam Gambar 2.1 *Cryptography Primitives Taxonomy*. Algoritma kunci simetrik atau yang lebih dikenal dengan algoritma kunci rahasia (*secret key*) merupakan algoritma yang menggunakan satu kunci yang sama untuk melakukan enkripsi dan dekripsi. Sebuah *plaintext* dapat dienkripsi menjadi *ciphertext* dengan algoritma kunci simetrik menggunakan *secret key*, yang hanya diketahui oleh pengirim dan penerima. *Secret key* ini dibuat secara terpisah dari *plaintext*. Saat *ciphertext* dikirimkan, dan kemudian diterima oleh penerima, maka *ciphertext* tersebut akan didekripsikan kembali menggunakan algoritma dekripsi dan menggunakan kunci yang sama.

Algoritma ini sangat sederhana dalam penerapannya, karena setiap pasang pengirim dan penerima menggunakan satu *secret key* saja. Dalam jaringan yang sangat besar, akan semakin banyak terdapat pengirim dan penerima. Hal ini dapat menjadi masalah karena *secret key management* yang digunakan akan semakin kompleks.

Algoritma kunci simetrik dibagi menjadi dua bagian, yaitu *block cipher* dan *stream cipher*. *Block cipher* adalah algoritma kunci simetrik yang mengenkripsi setiap blok dalam *plaintext* menjadi *ciphertext* dalam satu waktu. Sedangkan *stream cipher* adalah algoritma kunci simetrik yang mengenkripsi setiap bit dalam *plaintext* menjadi *ciphertext* dalam satu waktu. Salah satu contoh algoritma *stream cipher* ini adalah Grain yang akan dibahas pada bagian selanjutnya.

2.2 Aljabar Abstrak (*Abstract Algebra*)

2.2.1 Grup

Grup G adalah sebuah himpunan dengan operasi $a * b$, dengan elemen khusus e yang disebut identitas, dan memenuhi kriteria berikut:

- Hukum Identitas.

  $a * e = e * a = a$ (2.3)

  Untuk semua $a \in G$
- Hukum Invers.

  $a * b = b * a = e$ (2.4)

  Untuk semua $a \in G$ terdapat $b \in G$ (b adalah invers dari a)
- Hukum Asosiatif.

  $a * (b * c) = (a * b) * c$ (2.5)

  Untuk semua $a, b, c \in G$

Jika operasi $a * b$ adalah komutatif, yaitu $a * b = b * a$, maka grup tersebut disebut *Abelian Group*. Pada grup ini, operasi yang digunakan ditulis sebagai operasi penambahan (*addition*), sehingga elemen identitas grup adalah 0. Elemen invers dari grup ini ditulis $-a$.

Pada kasus yang lain, operasi yang digunakan adalah perkalian (*multiplication*), sehinggga elemen identitasnya adalah 1. Elemen invers dari grup ini ditulis a^{-1} [Gar-05].

2.2.2 *Ring*

Ring R adalah sebuah himpunan dengan dua operasi (+ dan *) dan dengan elemen khusus 0 yang disebut sebagai identitas penambahan, dan memenuhi kriteria berikut:

- R adalah *Abelian group* dengan operasi + dan identitas 0.
- Hukum Asosiatif.

  $a \cdot (b \cdot c) = (a \cdot b) \cdot c$ (2.6)

  $1 \cdot a = a \cdot 1 = a$ (2.7)

  Untuk semua $a, b, c \in R$

- Hukum Distributif.

  $a \cdot (b + c) = (a \cdot b) + (a \cdot c)$ (2.8)

  $(b + c) \cdot a = (b \cdot a) + (c \cdot a)$ (2.9)

  Untuk semua $a, b, c \in R$

Ring memiliki properti tambahan, yaitu:

- Jika terdapat elemen 1 pada *ring* yang memenuhi $1 \cdot a = a \cdot 1 = a$ untuk semua $a \in R$, maka 1 disebut identitas perkalian (*multiplicative identity*) atau disebut unit *ring*.
- Jika $a.b = b.a$ untuk semua $a, b \in R$ maka *ring* tersebut disebut *ring* komutatif (*commutative ring*). Oleh karena itu, sebuah *ring* disebut komutatf jika dan hanya jika operasi perkaliannya memenuhi hukum komutatif.
- Pada sebuah *ring*, untuk sebuah elemen $a \in R$, jika terdapat $a^{-1} \in R$, yang memenuhi $a * a^{-1} = a^{-1} * a$, maka a^{-1} disebut *invers* perkalian (*multiplicative inverse*) dari a. Himpunan seluruh *unit* pada *ring* yang dinotasikan $R^{\times}$ disebut *group unit* pada R [Gar-05].

2. 2.3 *Field*

Field F adalah sebuah himpunan dengan operasi + dan $\cdot$, dengan syarat

- $(F, +)$ adalah sebuah komutatif grup
- $(F^{x}, \bullet)$ adalah sebuah komutatif grup
- memenuhi hukum distributif.

Sehingga sebuah *field* adalah sebuah *ring* komutatif yang setiap elemen bukan nol (*non-zero element*) adalah sebuah *unit* [Gar-05].

2.2.4 *Finite Field*

Finite Field adalah sebuah *field* yang memiliki jumlah elemen terbatas (*finite*) yang dinotasikan dengan F_q atau $GF(q)$, dengan q adalah jumlah elemen [Gar-05].

2.2.5 *Polynomial Basis Representation*

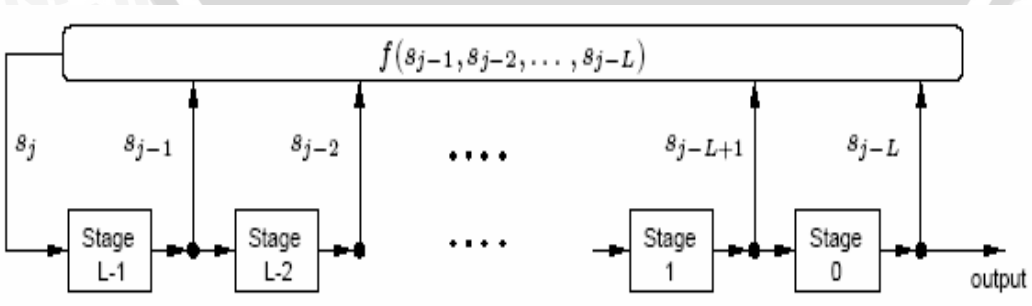
Finite Field GF(p^m) dapat dipandang sebagai ruang vektor dengan dimensi m. Representasi basis *field* yang digunakan adalah standar basis. Pada standar basis, elemen dari GF(p^m) direpresentasikan dengan kombinasi linear basis himpunan $\{1, \alpha, \alpha^2, ..., \alpha^{m-2}, \alpha^{m-1}\}$, dengan α adalah akar dari *irreducible polynomial* yang digunakan untuk membangun *field* $GF(p^m)$ dari *field* $GF(p)$. *Irreducible polynomial* $P(x)$ adalah sebuah polinomial yang tidak dapat difaktorkan menjadi polinomial $p_1(x)$ dan $p_2(x)$ dengan derajat yang lebih rendah.

Dengan penggunaan basis ini, maka seluruh perhitungan pada *field* ini, baik penjumlahan, pengurangan, perkalian maupun inversi harus direduksi dengan operasi *modulo* $P(x)$.

2.2.6 *Feedback Shift Register*

Sebuah *feedback shift register* dengan panjang L, yang tiap unitnya dapat menyimpan satu bit, memiliki satu input dan satu output, serta memiliki sebuah *clock control* yang mengatur perpindahan data. Dalam setiap *clock*, operasi yang terjadi adalah sebagai berikut:

a. isi dari *stage* 0 adalah output dan akan membentuk bagian dari *output sequence*
b. isi dari *stage* i bergerak ke *stage* $i-1$ untuk setiap *i*, $1 \le i \le L-1$
c. isi yang baru dari *stage* $L-1$ digunakan sebagai bit *feedback* $s_j = f(s_{j-1}, s_{j-2}, ..., s_{j-L})$ dengan *feedback function* f adalah *Boolean function* dan s_{j-i} adalah isi sebelumnya dari *stage* $L-i$, dengan $1 \le i \le L$.

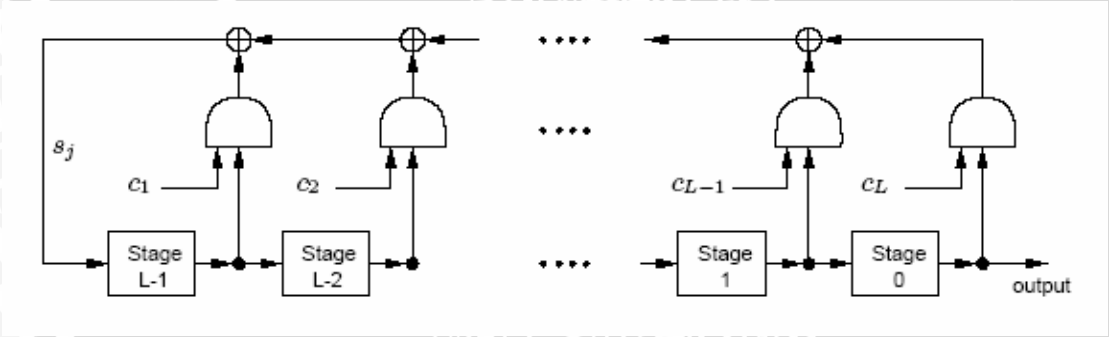


2.3 *Feedback Shift Register* dengan Panjang L

2.2.6.1 *Linear Feedback Shift Register*

Linear feedback shift register dengan panjang L terdiri dari L *stages* (atau elemen *delay*) dengan urutan mulai $0,1,...,L-1.$, yang tiap unitnya dapat menyimpan satu bit dan memiliki satu input dan satu output dan sebuah *clock control* yang mengatur perpindahan data. Dalam setiap *clock*, operasi yang terjadi adalah:

a. isi dari *stage* 0 adalah output dan akan membentuk bagian dari *output sequence*
b. isi dari *stage* i bergerak ke *stage* $i-1$ untuk setiap i, $1 \le i \le L-1$
c. isi yang baru dari *stage* $L-1$ digunakan sebagai bit *feedback* s_j dengan cara menambahkan dengan isi sebelumnya menggunakan operasi penambahan *modulo* 2 dengan *subset* yang sama sepanjang $0,1,...,L-1.$



Gambar 2.4 *Linear Feedback ShiftRregister* dengan Panjang L

LFSR dalam Gambar 2.4 dinotasikan dengan $\langle L, C(D) \rangle$. Dengan sebuah persamaan *connection polynomial*.

$$C(D) = 1 + c_1 D + c_2 D^2 + ... + c_L D^L \in Z_2[D] \qquad (2.10)$$

Sebuah *linear feedback shift register* dikatakan *nonsingular* jika *degree* dari $C(D)$ adalah L (sehingga, $c_L = 1$). Jika isi awal (*initial content*) dari *stage* i adalah $s_i \in \{0,1\}$ untuk setiap i, $1 \le i \le L-1$ maka $[s_{L-1},...,s_1,s_0]$ disebut *initial state linear feedback shift register*.

Jika *initial state linear feedback shift register* adalah $[s_{L-1},...,s_1,s_0]$ maka output *sequence* $s = s_0, s_1, s_2,...$ secara unik merupakan rekursi dari

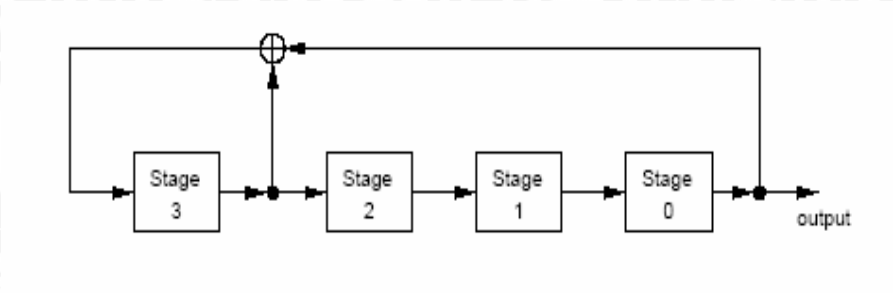
$$s_j = (c_1 s_{j-1} + c_2 s_{j-2} + ... + c_L s_{j-L}) \bmod 2 \text{ untuk } j \geq L \qquad (2.11)$$

Setiap *output sequence* untuk semua *initial states* yang mungkin dari sebuah *linear feedback shift register* $\langle L, C(D) \rangle$ adalah periodik, jika dan hanya jika *connection polynomial* $C(D)$ memiliki *degree* L. Sebuah *linear feedback shift register* $\langle L, C(D) \rangle$ disebut *singular* jika $C(D)$ memiliki *degree* kurang dari L $(c_L = 0)$.

Gambar 2.5 merupakan salah satu contoh dari *nonsingular linear feedback shift register* LFSR $\langle 4, 1 + D + D^4 \rangle$ dengan *connection polynomial* $C(D) = 1 + D + D^4$, dan *initial state* [0,1,1,0]. Isi masing-masing *stage* pada saat *t* dapat dilihat pada Tabel 2.1 berikut.

Tabel 2.1 Isi Tiap *Stage* LFSR [MOV-97]

t	Stage			
	3	2	1	0
0	0	1	1	0
1	0	0	1	1
2	1	0	0	1
3	0	1	0	0
4	0	0	1	0
5	0	0	0	1
6	1	0	0	0
7	1	1	0	0
8	1	1	1	0
9	1	1	1	1
10	0	1	1	1
11	1	0	1	1
12	0	1	0	1
13	1	0	1	0
14	1	1	0	1
15	0	1	1	0



Gambar 2.5 LFSR <4, 1+ D + D^4>

Output sequence s adalah 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, … dengan periode 15.

2.2.6.2 *Non Linear Feedback Shift Register*

Non linear feedback shift register adalah sebuah *shift register* yang menggunakan *feedback* sebagai nilai kembalian setiap *clock*-nya. *Feedback* dalam *non linear feedback shift register* ini adalah sebuah *polynomial* yang *non linear*.

2.3 *Randomness Test*

Untuk menguji tingkat *randomness* suatu *sequences*, dapat digunakan *randomness* test yang merupakan standar dari *National Institute of Standard and Technology* (NIST), dengan menggunakan parameter *P-value* atau yang biasa disebut "*tail probability*". *P-value* adalah probabilitas yang dipilih sebagai nilai acuan untuk menentukan tingkat keacakan dalam *null hypothesis. Null hypothesis* merupakan kondisi atau properti *default* dari *sequence* yang diobservasi.

Nilai *P-value* digunakan untuk menentukan tingkat keacakan suatu *sequence*. Jika nilai *P-value* lebih dari atau sama dengan 0,01, maka *sequence* tersebut dikatakan *random*. Dan sebaliknya, suatu *sequence* dikatakan tidak *random* jika nilai *P-value* kurang dari 0,01 [NIS-01].

Test standar NIST yang digunakan adalah *Frequency Test, Frequency Test within a Block, Runs Test, The Longest Run in a Block* dan *Binary Matrix Test.*

2.3.1 Frequency test

Tujuan dari test ini adalah untuk menguji apakah dalam satu periode *sequence* jumlah antara nol (*zero*) dan satu (*one*) seimbang. Langkah-langkah yang digunakan untuk menjalankan test ini adalah :

1. Mengkonversikan nilai *sequence* menjadi nilai +1 dan -1 masing-masing untuk nilai nol dan satu dalam sebuah *sequence* ε_i dengan panjang n untuk menghasilkan

   $$S_n = X_1 + X_2 + ... + X_n \quad (2.12)$$

   dengan $X_i = 2\varepsilon_i - 1$. (2.13)

2. Menghitung nilai statistik test dengan menggunakan persamaan

   $$S_{obs} = \frac{|S_n|}{\sqrt{n}} \quad (2.14)$$

3. Menghitung nilai *P-value* dengan menggunakan *complementary error function*.

   $$P - value = erfc\left(\frac{S_{obs}}{\sqrt{2}}\right) \quad (2.15)$$

4. Menentukan keacakan *sequence* yang diuji.

2.3.2 *Frequency Test within a Block*

Tujuan dari test ini adalah untuk menentukan apakah frekuensi dari satu (*one*) dalam sebuah $M - bit$ blok mendekati $M/2$, sebagai referensi tingkat keacakan. Untuk menjalankan test ini, digunakan langkah-langkah berikut ini:

1. *Sequence* n bit yang diuji dibagi menjadi beberapa $N = \left\lfloor \frac{n}{M} \right\rfloor$ bagian blok yang tidak *overlapping*.
2. Menghitung proporsi nilai π_i dalam setiap $M - bit$ blok dalam *sequence* ε_i dengan persamaan

   $$\pi_i = \frac{\sum_{j=1}^{M} \varepsilon_{(i-1)M+j}}{M} \text{ dengan } 1 \le i \le N \quad (2.16)$$

3. Menghitung nilai *chi-square* dengan persamaan

   $$\chi^2(obs) = 4M \sum_{i=1}^{N} (\pi_i - 1/2)^2 \quad (2.17)$$

4. Menghitung nilai *P-value* dengan *incomplete gamma function*.

$$P-value = igamc(N/2, \chi^2(obs)/2) \quad (2.18)$$

5. Menentukan keacakan *sequence* yang diuji.

2.3.3 Runs Test

Tujuan utama dari *runs test* ini adalah untuk mengetahui jumlah *run* dalam satu periode *sequence*, dan untuk melihat kecepatan osilasi antara nol (*zero*) dan satu (*one*). Untuk menghitung tingkat keacakan suatu *sequence* dengan menggunakan test ini, maka dapat dilakukan langkah-langkah sebagai berikut :

1. Menghitung nilai proporsi π_i dalam *sequence* ε_i dengan menggunakan persamaan berikut

$$\pi = \frac{\sum \varepsilon_i}{n} \quad (2.19)$$

2. Menghitung nilai persyaratan untuk menjalankan test ini dengan menggunakan test sebelumnya, yaitu *frequency test*. Jika telah memenuhi persyaratan *frequency test* tersebut, maka *runs test* ini dapat dijalankan. Jika tidak maka dipastikan *sequence* yang diuji tidak random.
3. Menghitung statistik test dengan menggunakan persamaan

$$V_n(obs) = \sum_{k=1}^{n-1} r(k) + 1 \quad (2.20)$$

Dengan nilai

$$r(k)\begin{cases} 1 & \varepsilon_k = \varepsilon_{k+1} \\ 0 & \varepsilon_k \neq \varepsilon_{k+1} \end{cases}$$

4. Menghitung nilai *P-value* dengan *complementary error function*.

$$P-value = erfc\left(\frac{|V_n(obs) - 2n\pi(1-\pi)|}{2\sqrt{2n}\pi(1-\pi)}\right) \quad (2.21)$$

5. Menentukan keacakan *sequence* yang diuji.

2.3.4 *Test for the Longest Run of Ones in a Block*

Test ini digunakan untuk menghitung *run* suatu *sequence* dalam sebuah blok. Panjang blok yang dianjurkan dalam test ini dapat dilihat pada Tabel 2.2.

Tabel 2.2 Panjang Bit dalam Satu Blok [NIS-01]

Minimum *n*	***M***
128	8
6272	128
750.000	10^4

Langkah-langkah yang digunakan untuk menjalankan test ini adalah sebagai berikut:

1. *Sequence* yang diuji dibagi menjadi beberapa $M-bit$ blok
2. Menabulasikan frekuensi v_i dari *run* satu (*one*) terpanjang dalam setiap blok. Untuk nilai M yang diberikan, nilai v_i dapat dilihat pada Tabel 2.3.

Tabel 2.3 Nilai v_i dalam Tiap Blok [NIS-01]

v_i	$M=8$	$M=128$	$M=10^4$
v_0	≤ 1	≤ 4	≤ 10
v_1	2	5	11
v_2	3	6	12
v_3	≥ 4	7	13
v_4		8	14
v_5		≥ 9	15
v_6			≥ 16

3. Menghitung nilai statistik test dengan persamaan berikut

$$\chi^2(obs) = \sum_{i=0}^{K} \frac{(v_i - N\pi_i)^2}{N\pi_i} \qquad (2.22)$$

Dengan nilai K dan N sesuai dengan nilai M yang diberikan seperti pada 2.4

Tabel 2.4 Nilai K dan N untuk Setiap M [NIS-01]

M	***K***	***N***
8	3	16
128	5	49
10^4	6	75

4. Menghitung nilai *P-value* dengan *incomplete gamma function*.

$$P-value = igamc(\frac{K}{2}, \frac{\chi^2(obs)}{2}) \quad (2.23)$$

5. Menentukan keacakan *sequence* yang diuji.

2.3.5 Binary Matrix Test

Tujuan dari test ini adalah untuk mengetahui ketergantungan *linear* dari setiap blok *sequence* dengan cara mengetahui *rank* dari sub-matrik dari seluruh *sequence*. Langkah-langkah untuk menjalankan test ini adalah :

1. Membagi *sequence* menjadi $M \cdot Q$ matrik-matrik sebanyak $N = \left\lfloor \frac{n}{MQ} \right\rfloor$ blok.
2. Menghitung *rank* tiap matrik dengan algoritma berikut:

   a. *Forward Application of Elementary Row Operations*

   Matrik dengan dimensi $m \times m$ dinotasikan $a_{i,j}$

   1. Menentukan $i = 1$
   2. Jika elemen $a_{i,i} = 0$, maka semua elemen pada baris ke i ditukar dengan seluruh elemen pada baris berikutnya yang mengandung nilai 1 pada kolom ke i. Jika tidak ada nilai 1 pada posisi ini, maka dilanjutkan ke langkah empat.

3. Jika elemen $a_{i,i} = 1$, maka jika ada baris yang mengandung nilai 1 pada kolom ke i, maka setiap elemen pada baris tersebut diganti dengan cara meng-XOR-kan elemen tersebut dengan elemen pada baris ke i.
   a. Menentukan $baris = i + 1$
   b. Menentukan $kolom = i$.
   c. Jika $a_{baris,kolom} = 0$, maka dilanjutkan ke langkah 3g.
   d. $a_{baris,kolom} = a_{baris,kolom} \oplus a_{i,kolom}$
   e. Jika $kolom = m$, maka dilanjutkan ke langkah 3g.
   f. $kolom = kolom + 1$, dilanjutkan ke langkah 3d.
   g. Jika $baris = m$, maka dilanjutkan ke langkah 4.
   h. $baris = baris + 1$, dilanjutkan ke langkah 3b.
4. Jika $i < m - 1$, maka $i = i + 1$, dan dilanjutkan ke langkah 2
5. Operasi *Forward Row* selesai.

b. Backward Application of Elementary Row Operations

1. Menentukan $i = m$.
2. Jika elemen $a_{i,i} = 0$, maka semua elemen pada baris ke i ditukar dengan seluruh elemen pada baris berikutnya yang mengandung nilai 1 pada kolom ke i. Jika tidak ada nilai 1 pada posisi ini, maka dilanjutkan ke langkah empat.
3. Jika elemen $a_{i,i} = 1$, maka jika ada baris yang mengandung nilai 1 pada kolom ke i, maka setiap elemen pada baris tersebut diganti dengan cara meng-XOR-kan elemen tersebut dengan elemen pada baris ke i.
   a. Menentukan $baris = i - 1$
   b. Menentukan $kolom = i$
   c. Jika $a_{baris,kolom} = 0$, maka dilanjutkan ke langkah 3g.
   d. $a_{baris,kolom} = a_{baris,kolom} \oplus a_{i,kolom}$
   e. Jika $kolom = 1$, maka dilanjutkan ke langkah 3g.
   f. $kolom = kolom - 1$, maka dilanjutkan ke langkah 3d.

g. Jika $baris = 1$, maka dilanjutkan ke langkah 4.

h. $baris = baris - 1$, maka dilanjutkan ke langkah 3b.

4. Jika $i > 2$, maka $i = i - 1$
5. Operasi *Backward row* selesai.
6. *Rank* matriks adalah jumlah baris yang mengandung nilai 1.

3. Menentukan nilai masing-masing variabel berikut:

a. F_M adalah jumlah matrik dengan $R_\ell = M$

b. F_{M-1} adalah jumlah matrik dengan $R_\ell = M - 1$

c. $N - F_M - F_{M-1}$ adalah jumlah matrik yang tersisa

4. Menghitung nilai statistik test dengan persamaan berikut

$$\chi^2(obs) = \frac{(F_M - 0{,}2888N)^2}{0{,}2888N} + \frac{(F_{M-1} - 0{,}5776N)^2}{0{,}5776N} + \frac{(N - F_M - F_{M-1} - 0{,}1336N)^2}{0{,}1336N} \quad (2.24)$$

5. Menghitung nilai *P-value* dengan *incomplete gamma function*.

$$P - value = igamc(1, \frac{\chi^2(obs)}{2}) \quad (2.25)$$

6. Menentukan keacakan *sequence* yang diuji.

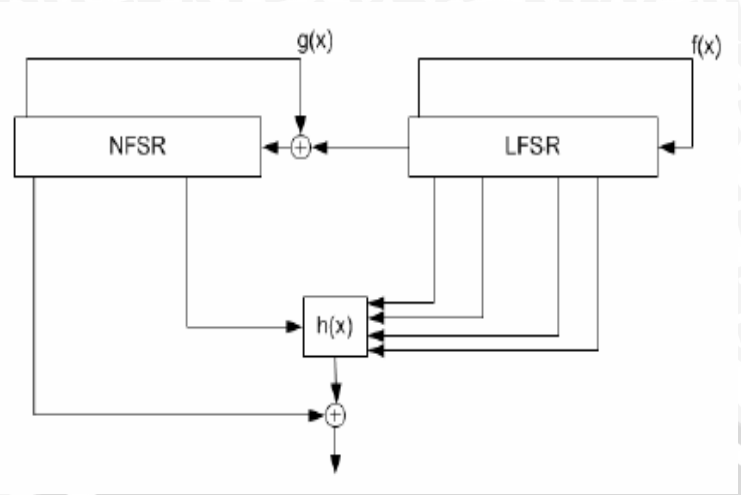
2.4 Grain

Pada bagian ini akan dibahas tentang sejarah Grain, arsitektur Grain dan *keystream* generator.

2.4.1 Sejarah Grain

Grain merupakan *stream cipher* yang dibuat oleh Martin Hell, Thomas Johansson, dan Will Meier dari Lund University, Swedia, untuk diikutsertakan dalam proposal eSTREAM sebuah proyek *security* Eropa.

Grain diharapkan akan menggantikan A5/1 pada GSM *System* yang telah dipecahkan oleh Alex Biryukov, Adi Shamir dan David Wagner dalam waktu kurang dari satu detik dengan menggunakan PC 128 MB RAM [EJ-02]. Biham dan Dunkelman juga telah berhasil memecahkan A5/1 pada tahun 2000 [EJ-02].



Gambar 2.6 Grain *Cipher* [HJM-05]

Grain *stream cipher* merupakan tipe *binary additive stream cipher*, kunci *k* dan parameter tambahan yaitu *Initialization Vector* (IV) digunakan untuk menghasilkan sebuah *binary sequence* yang disebut *keystream*.

2.4.2 Algoritma Grain

Arsitektur Grain terdiri dari tiga kelompok besar, yaitu LFSR, NFSR dan *Non linear Filter*. LFSR ini terdiri 80 bit yang dilambangkan dengan $s_i, s_{i+1}, s_{i+2}, ..., s_{i+79}$, sedangkan NFSR dilambangkan dengan $b_i, b_{i+1}, b_{i+2}, ...b_{i+79}$ untuk Grain v0 dan Grain v1.

Sedangkan untuk Grain 128, panjang LFSR dan NFSR masing-masing adalah 128 bit. LFSR ini dilambangkan dengan $s_i, s_{i+1}, s_{i+2}, ..., s_{i+127}$, dan NFSR dilambangkan dengan $b_i, b_{i+1}, b_{i+2}, ...b_{i+127}$.

2.4.2.1 Grain v0

Pada Grain v0 terdiri atas tiga bagian yaitu, yaitu LFSR, NFSR dan *non linear filter***.**

a. LFSR

Fungsi polinomial untuk nilai kembalian LFSR adalah f(x) yang merupakan *primitive polynomial* dengan derajat 80.

$$f(x) = 1 + x^{18} + x^{29} + x^{42} + x^{57} + x^{67} + x^{80} \quad (2.26)$$

b. NFSR

Fungsi g(x) adalah nilai kembalian NFSR dengan persamaan

$$\begin{aligned} g(x) = 1 + x^{17} + x^{20} + x^{28} + x^{35} + x^{43} + x^{47} + x^{52} + x^{59} \\ + x^{65} + x^{71} + x^{80} + x^{17}x^{20} + x^{43}x^{47} + x^{65}x^{71} + x^{20}x^{28}x^{35} \\ + x^{47}x^{52}x^{59} + x^{17}x^{35}x^{52}x^{71} + x^{20}x^{28}x^{43}x^{47} + x^{17}x^{20}x^{59}x^{65} \\ + x^{17}x^{20}x^{28}x^{35}x^{43} + x^{47}x^{52}x^{59}x^{65}x^{71} + x^{28}x^{35}x^{43}x^{47}x^{52}x^{59} \end{aligned} \quad (2.27)$$

c. *Non Linear Filter*

Fungsi *Non linear filter* h(x) ini diambil dari masing-masing state *linear feedback shift register* dan *non linear feedback shift register* dengan persamaan

$$\begin{aligned} h(x) = x_1 + x_4 + x_0x_3 + x_2x_3 + x_3x_4 + x_0x_1x_2 + x_0x_2x_3 \\ + x_0x_2x_4 + x_1x_2x_4 + x_2x_3x_4 \end{aligned} \quad (2.28)$$

dengan nilai x_0, x_1, x_2, x_3, x_4 masing-masing adalah s_{i+3}, s_{i+25}, s_{i+46}, s_{i+64}, dan b_{i+63}. Output dari *filter non linear* ini akan di-*masked* dengan bit dari *non linear feedback shift register* untuk menghasilkan *keystream*.

Untuk algoritma Grain v0 ini ditunjukkan dalam Gambar 2.7.

Algoritma Grain v0

INPUT : 80 bit kunci (*key*)
64 bit *initialization vector* (*iv*)
OUTPUT : 80 bit *keystream*

1. For *i* from 0 to 79 do
   1.1 $b_i \leftarrow key$
   1.2 $s_i \leftarrow iv$
   1.3 if $63 < i < 80$
   $s_i \leftarrow 1$

2. For *i* from 1 to 160 do
   2.1 $f(x) \leftarrow s_{62} + s_{51} + s_{38} + s_{23} + s_{13} + s_0$
   2.2 $g(x) \leftarrow s_0 + b_{63} + b_{60} + b_{52} + b_{45} + b_{37} + b_{33} + b_{28} + b_{21} + b_{15} + b_9 + b_0$
   $b_{63}b_{60} + b_{37}b_{33} + b_9b_{15} + b_{60}b_{52}b_{45} + b_{33}b_{28}b_{21} + b_{63}b_{45}b_{28}b_9$
   $b_{60}b_{52}b_{37}b_{33} + b_{63}b_{60}b_{21}b_{15} + b_{63}b_{60}b_{52}b_{45}b_{37}$
   $b_{33}b_{28}b_{21}b_{15}b_9 + b_{52}b_{45}b_{37}b_{33}b_{28}b_{21}$
   2.3 $h(x) \leftarrow s_{25} + b_{63} + s_3s_{64} + s_{46}s_{64} + s_{64}b_{63} + s_3s_{25}s_{46} + s_3s_{46}s_{64} + s_3s_{46}b_{63}$
   $s_{25}s_{46}b_{63} + s_{46}s_{64}b_{63}$
   2.4 $output \leftarrow h(x) \wedge b_0$
   2.5 $f(x) \leftarrow f(x) \wedge output$
   2.6 $g(x) \leftarrow g(x) \wedge output$

3. For *i* from 0 to 79 do
   3.1 $f(x) \leftarrow s_{62} + s_{51} + s_{38} + s_{23} + s_{13} + s_0$
   3.2 $g(x) \leftarrow s_0 + b_{63} + b_{60} + b_{52} + b_{45} + b_{37} + b_{33} + b_{28} + b_{21} + b_{15} + b_9 + b_0$
   $b_{63}b_{60} + b_{37}b_{33} + b_9b_{15} + b_{60}b_{52}b_{45} + b_{33}b_{28}b_{21} + b_{63}b_{45}b_{28}b_9$
   $b_{60}b_{52}b_{37}b_{33} + b_{63}b_{60}b_{21}b_{15} + b_{63}b_{60}b_{52}b_{45}b_{37}$
   $b_{33}b_{28}b_{21}b_{15}b_9 + b_{52}b_{45}b_{37}b_{33}b_{28}b_{21}$
   3.3 $h(x) \leftarrow s_{25} + b_{63} + s_3s_{64} + s_{46}s_{64} + s_{64}b_{63} + s_3s_{25}s_{46} + s_3s_{46}s_{64} + s_3s_{46}b_{63}$
   $s_{25}s_{46}b_{63} + s_{46}s_{64}b_{63}$
   3.4 $output \leftarrow h(x) \wedge b_0$
   3.5 $keystream \leftarrow output$
   3.6 $return\ (keystream)$

Gambar 2.7 Algoritma Grain v0

Pada algoritma Grain v0 ini, masukan 80 bit *key* dan 64 bit *initialization vector* akan diproses dalam LFSR dan NFSR dengan persamaan f(x) dan g(x). *Keystream* sebanyak 80 bit akan dihasilkan dari persamaan h(x).

2.4.2.2 Grain v1

Pada Grain v0 terdiri atas tiga bagian yaitu, yaitu LFSR, NFSR dan *non linear filter*.

a. LFSR

Fungsi kembalian LFSR adalah f(x) merupakan *primitive polynomial* dengan derajat 80.

$$f(x) = 1 + x^{18} + x^{29} + x^{42} + x^{57} + x^{67} + x^{80} \tag{2.29}$$

b. NFSR

Fungsi polinomial untuk nilai kembalian NFSR adalah fungsi g(x) dengan persamaan

$$\begin{aligned} g(x) = 1 &+ x^{18} + x^{20} + x^{28} + x^{35} + x^{43} + x^{47} + x^{52} + x^{59} \\ &+ x^{66} + x^{71} + x^{80} + x^{17}x^{20} + x^{43}x^{47} + x^{65}x^{71} + x^{20}x^{28}x^{35} \\ &+ x^{47}x^{52}x^{59} + x^{17}x^{35}x^{52}x^{71} + x^{20}x^{28}x^{43}x^{47} + x^{17}x^{20}x^{59}x^{65} \\ &+ x^{17}x^{20}x^{28}x^{35}x^{43} + x^{47}x^{52}x^{59}x^{65}x^{71} + x^{28}x^{35}x^{43}x^{47}x^{52}x^{59} \end{aligned} \tag{2.30}$$

c. *Non Linear Filter*

Non linear filter ini berfungsi sebagai *filter* dengan fungsi *non linear* h(x) yang variabelnya berasal dari *linear feedback shift register* dan *non linear feedback shift register*. Fungsi *non linear* tersebut adalah

$$\begin{aligned} h(x) = {} & x_1 + x_4 + x_0x_3 + x_2x_3 + x_3x_4 + x_0x_1x_2 + x_0x_2x_3 \\ & + x_0x_2x_4 + x_1x_2x_4 + x_2x_3x_4 \end{aligned} \tag{2.31}$$

dengan nilai x_0, x_1, x_2, x_3, x_4 masing-masing adalah s_{i+3}, s_{i+25}, s_{i+46}, s_{i+64}, and b_{i+63}. Output dari *non linear filter* ini akan di-*masked* dengan bit dari *non linear feedback shift register* untuk menghasilkan *keystream,* dengan persamaan

$$z_i = \sum_{k \in A} b_{i+k} + h(s_{i+3}, s_{i+25}, s_{i+46}, s_{i+64}, s_{i+63})$$

dengan $A = \{1,2,4,10,31,43,56\}$ (2.31)

Algoritma Grain v1 secara lebih jelas dapat dilihat dalam Gambar 2.8.

Algoritma Grain v1

INPUT : 80 bit kunci (*key*)
64 bit *initialization vector* (*iv*)
OUTPUT : 80 bit *keystream*

---

1. For *i* from 0 to 79 do
   - 1.1 $b_i \leftarrow key$
   - 1.2 $s_i \leftarrow iv$
   - 1.3 if $63 < i < 80$
     $s_i \leftarrow 1$

2. For *i* from 1 to 160 do
   - 2.1 $f(x) \leftarrow s_{62} + s_{51} + s_{38} + s_{23} + s_{13} + s_0$
   - 2.2 $g(x) \leftarrow s_0 + b_{62} + b_{60} + b_{52} + b_{45} + b_{37} + b_{33} + b_{28} + b_{21} + b_{14} + b_9 + b_0$
     $b_{63}b_{60} + b_{37}b_{33} + b_9b_{15} + b_{60}b_{52}b_{45} + b_{33}b_{28}b_{21} + b_{63}b_{45}b_{28}b_9$
     $b_{60}b_{52}b_{37}b_{33} + b_{63}b_{60}b_{21}b_{15} + b_{63}b_{60}b_{52}b_{45}b_{37}$
     $b_{33}b_{28}b_{21}b_{15}b_9 + b_{52}b_{45}b_{37}b_{33}b_{28}b_{21}$
   - 2.3 $h(x) \leftarrow s_{25} + b_{63} + s_3s_{64} + s_{46}s_{64} + s_{64}b_{63} + s_3s_{25}s_{46} + s_3s_{46}s_{64} + s_3s_{46}b_{63}$
     $s_{25}s_{46}b_{63} + s_{46}s_{64}b_{63}$
   - 2.4 $output \leftarrow h(x) \hat{} b_1 \hat{} b_2 \hat{} b_4 \hat{} b_{10} \hat{} b_{31} \hat{} b_{43} \hat{} b_{56}$
   - 2.5 $f(x) \leftarrow f(x) \hat{} output$
   - 2.6 $g(x) \leftarrow g(x) \hat{} output$

3. For *i* from 0 to 79 do
   - 3.1 $f(x) \leftarrow s_{62} + s_{51} + s_{38} + s_{23} + s_{13} + s_0$
   - 3.2 $g(x) \leftarrow s_0 + b_{62} + b_{60} + b_{52} + b_{45} + b_{37} + b_{33} + b_{28} + b_{21} + b_{14} + b_9 + b_0$
     $b_{63}b_{60} + b_{37}b_{33} + b_9b_{15} + b_{60}b_{52}b_{45} + b_{33}b_{28}b_{21} + b_{63}b_{45}b_{28}b_9$
     $b_{60}b_{52}b_{37}b_{33} + b_{63}b_{60}b_{21}b_{15} + b_{63}b_{60}b_{52}b_{45}b_{37}$
     $b_{33}b_{28}b_{21}b_{15}b_9 + b_{52}b_{45}b_{37}b_{33}b_{28}b_{21}$
   - 3.3 $h(x) \leftarrow s_{25} + b_{63} + s_3s_{64} + s_{46}s_{64} + s_{64}b_{63} + s_3s_{25}s_{46} + s_3s_{46}s_{64} + s_3s_{46}b_{63}$
     $s_{25}s_{46}b_{63} + s_{46}s_{64}b_{63}$
   - 3.4 $output \leftarrow h(x) \hat{} b_1 \hat{} b_2 \hat{} b_4 \hat{} b_{10} \hat{} b_{31} \hat{} b_{43} \hat{} b_{56}$
   - 3.5 $keystream \leftarrow output$
   - 3.6 $return\ (keystream)$

Gambar 2.8 Algoritma Grain v1

Pada algoritma Grain v1, masukan 80 bit *key* dan 64 bit *initialization vector* akan diproses dalam LFSR dan NFSR dengan persamaan f(x) dan g(x). *Keystream* sebanyak 80 bit akan dihasilkan dari persamaan h(x).

2.4.2.3 Grain 128

Pada Grain 128 terdiri atas tiga bagian yaitu, yaitu LFSR, NFSR dan *non linear filter*.

a. LFSR

Nilai kembalian LFSR ini adalah fungsi f(x) yang merupakan *primitive polynomial* dengan derajat 80.

$$f(x) = 1 + x^{32} + x^{47} + x^{58} + x^{90} + x^{121} + x^{128} \quad (2.32)$$

b. NFSR

Fungsi kembalian NFSR adalah g(x) dengan persamaan

$$g(x) = 1 + x^{32} + x^{37} + x^{72} + x^{102} + x^{128} + x^{44}x^{60} + x^{61}x^{125} + x^{63}x^{67} + x^{69}x^{101} + x^{80}x^{88} + x^{110}x^{111} + +x^{115}x^{117} \quad (2.33)$$

c. *Non Linear Filter*

Fungsi h(x) adalah fungsi *non linear filter* ini diambil dari masing-masing state *linear feedback shift register* dan *non linear feedback shift register* dengan persamaan

$$h(x) = x_0x_1 + x_2x_3 + x_4x_5 + x_6x_7 + x_0x_4x_8 \quad (2.34)$$

dengan nilai $x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8$ masing-masing adalah

$b_{i+12}, s_{i+8}, s_{i+13}, s_{i+20}, b_{i+95}, s_{i+42}, s_{i+60}, s_{i+79}, s_{i+95}$

Output dari *filter non linear* ini akan di-*masked* dengan bit dari *non linear feedback shift register* dan *linear feedback shift register* untuk menghasilkan

keystream, dengan persamaan

$$z_i = \sum_{j \in A} b_{i+j} + h(x) + s_{i+93}$$

dengan $A = \{2,15,36,45,64,73,89\}$ (2.35)

Untuk penjelasan algoritma Grain 128 dapat dilihat dalam Gambar 2.9.

Algoritma Grain 128

INPUT : 128 bit kunci (*key*)
96 bit *initialization vector* (*iv*)
OUTPUT : 128 bit *keystream*

1. For *i* from 0 to 127 do
   - 1.1 $b_i \leftarrow key$
   - 1.2 $s_i \leftarrow iv$
   - 1.3 if $95 < i < 128$
     $s_i \leftarrow 1$

2. For *i* from 1 to 256 do
   - 2.1 $f(x) \leftarrow s_{96} + s_{81} + s_{70} + s_{38} + s_7 + s_0$
   - 2.2 $g(x) \leftarrow s_0 + b_{26} + b_{56} + b_{91} + b_{96} + b_0 + b_3 b_{67} + b_{11} b_{13}$
     $b_{17} b_{18} + b_{27} b_{59} + b_{40} b_{48} + b_{61} b_{65} + b_{68} b_{84}$
   - 2.3 $h(x) \leftarrow b_{12} s_8 + s_{13} s_{20} + b_{95} s_{42} + s_{60} s_{79} + b_{12} b_{95} s_{95}$
   - 2.4 $output \leftarrow h(x) \hat{} b_2 \hat{} b_{15} \hat{} b_{36} \hat{} b_{45} \hat{} b_{64} \hat{} b_{73} \hat{} b_{89} \hat{} s_{93}$
   - 2.5 $f(x) \leftarrow f(x) \hat{} output$
   - 2.6 $g(x) \leftarrow g(x) \hat{} output$

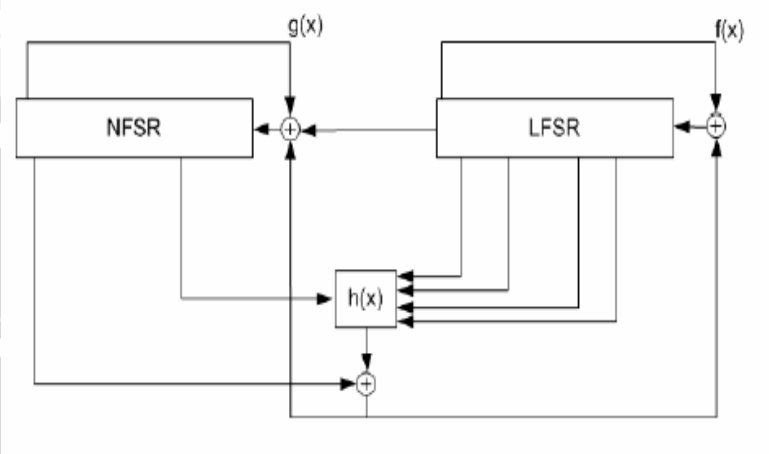
3. For *i* from 0 to 127 do
   - 3.1 $f(x) \leftarrow s_{96} + s_{81} + s_{70} + s_{38} + s_7 + s_0$
   - 3.2 $g(x) \leftarrow s_0 + b_{26} + b_{56} + b_{91} + b_{96} + b_0 + b_3 b_{67} + b_{11} b_{13}$
     $b_{17} b_{18} + b_{27} b_{59} + b_{40} b_{48} + b_{61} b_{65} + b_{68} b_{84}$
   - 3.3 $h(x) \leftarrow b_{12} s_8 + s_{13} s_{20} + b_{95} s_{42} + s_{60} s_{79} + b_{12} b_{95} s_{95}$
   - 3.4 $output \leftarrow h(x) \hat{} b_2 \hat{} b_{15} \hat{} b_{36} \hat{} b_{45} \hat{} b_{64} \hat{} b_{73} \hat{} b_{89} \hat{} s_{93}$
   - 3.5 $keystream \leftarrow output$
   - 3.6 $return\ (keystream)$

Gambar 2.9 Algoritma Grain 128

Pada algoritma Grain 128 ini, masukan 128 bit *key* dan 96 bit *initialization vector* akan diproses dalam LFSR dan NFSR dengan persamaan f(x) dan g(x). Dari persamaan h(x) akan dihasilkan *keystream* sebanyak 128 bit.

2.4.3 Keystream Generator

Sebelum suatu *stream cipher* dapat menghasilkan *keystream,* maka *stream cipher* tersebut harus diinisialisasi dengan *initialization vector* dan kunci. Untuk Grain v0 dan Grain v1, *initialization vector* ini terdiri dari 64 bit untuk mengisi LFSR, sedangkan 80 bit kunci akan digunakan untuk mengisi NFSR. Pada tahap inisialisasi kunci ini, *cipher* akan di-*clock* atau dijalankan sebanyak 160 kali tanpa menghasilkan *keystream*.



Gambar 2.10 Inisialisasi Kunci [HJM-05]

Untuk Grain 128, *cipher* ini akan di-*clock* atau dijalankan sebanyak 256 kali sebelum menghasilkan *keystream*, dengan 96 bit untuk mengisi LFSR, dan 128 bit kunci akan digunakan untuk mengisi NFSR.

2. 5 VHDL

Pada bagian ini akan dibahas tentang sejarah VHDL dan proses perancangan IC (*Integrated Circuit*).

2.5.1 Sejarah VHDL

Very High Speed Integated Circuit (VHSIC) *Hardware Description Language* (VHDL) pertama kali dikembangkan pada tahun 1982 oleh Departemen Pertahanan Amerika Serikat. VHDL dapat digunakan untuk mendeskripsikan fungsi, perilaku masukan dan keluaran dari rancangan rangkaian digital. VHDL digunakan sebagai standar HDL oleh IEEE yang dikenal sebagai IEEE Standard 1076-1987 yang

kemudian di-*update* pada tahun 1994, yaitu IEEE 1076-1993. Sedangkan oleh Departemen Pertahanan Amerika Serikat dikenal sebagai MIL-STD-454L.

Sintaks VHDL secara umum dapat dibagi menjadi tiga bagian utama, yaitu bagian *library*, bagian *entity* dan *architecture*, seperti dalam Gambar 2.8. Pada bagian *library*, digunakan untuk mendefinisikan *library* yang diperlukan oleh bagian *architecture*. Bagian *entity* digunakan untuk mendefinisikan pin-pin apa sajakah yang akan digunakan sebagai *input* dan *output*. Bagian *architecture*, digunakan untuk mendefinisikan detail arsitektur program yang dibuat.

```
library <Library_name>;
use.<Library_name>;
use.<Library_name>;
...

entity <Entity_name> is
   Port (
         <port_name> : <port_type>
         <port_name> : <port_type>
         ...
         );
End <Entity_name>;

architecture <Architecture_name> of <Entity_name_name> is
local_declaration;
local_declaration;
...

begin
program_statement;
program_statement;
...

end <Architecture_name >;
```

Gambar 2.11 Sintaks VHDL

2.5.2 Proses Perancangan IC (IC Design)

Proses perancangan IC menggunakan VHDL dilakukan dalam beberapa tahapan [Xil-06]:

1. Spesifikasi

   Pada tahap ini, penggunaan bahasa tingkat tinggi, VHDL misalnya, digunakan untuk mendeskripsikan *behavioral* desain rangkaian dalam sebuah *file* text.

2. Sintesis

   Proses sintesis ini mengkonversi desain *behavioral* menjadi gerbang-gerbang logika berdasar *file* yang telah dideskripsikan sebelumnya. Karena hasil dari proses ini berupa *netlist* yang sangat bergantung pada *vendor* dan spesifikasi *device-family*, maka penggunaan *library* harus disesuaikan dengan *vendor*.

3. Simulasi

   Desain rangkaian *programmable logic* diverifikasi dengan menggunakan simulator, yaitu sebuah *software* yang mengkonfirmasi fungsi dan *timing* sebuah rangkaian. Pada tahap simulasi ini dilakukan pengecekan kombinasi hasil dari desain rancangan digital.

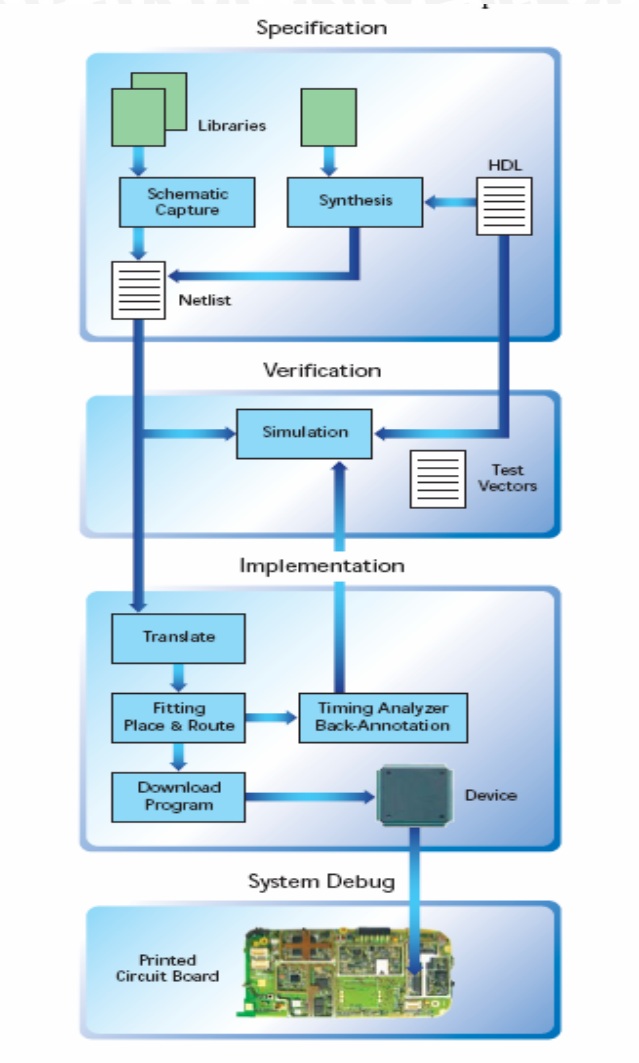
4. Implementasi dan Analisa

   Netlist sebuah desain menjelaskan gerbang logika yang digunakan dalam desain untuk *vendor* ataupun *device-family* tertentu. Setelah tahap simulasi, tahap berikutnya adalah pengimplementasian desain pada sebuah *chip*. Untuk CPLD (*Complex Programmable Logic Design*) tahap ini disebut *fitting*. Untuk FPGA (*Field Programmable Gate Arrays*) tahap ini disebut *place and route*. *Place* adalah sebuah proses untuk memilih modul yang spesifik, atau *logic stream* pada FPGA tempat gerbang logika akan ditempatkan. *Route* adalah proses *routing* fisik yang menghubungkan antar *logic stream*.

5. *Download* atau *Programming*

   Pada proses ini, konfigurasi informasi sebuah program di-*download* pada memori alat tersebut. Secara umum proses *download* ini mengacu pada *volatile device* misalnya SRAM FPGA.

Diagram alir proses di atas untuk *programable logic design* dapat dilihat dalam Gambar 2.9.



Gambar 2.12 Diagram Alir *Programable Logic Design* [Xil-06]

2.6 Perancangan Perangkat Lunak

Perancangan perangkat lunak atau biasanya berhubungan dengan perancangan sistem merupakan suatu fase yang sangat penting. Ada beberapa alat bantu yang dapat digunakan untuk perancangan sistem, antara lain Diagram Konteks dan *Data Flow Diagram*.

2.6.1 Diagram Konteks

Diagram konteks adalah sebuah diagram yang menggambarkan keseluruhan sistem, hubungan antara entitas luar, masukan dan keluaran dari sistem. Diagram konteks direpresentasikan dengan lingkaran tunggal yang mewakili keseluruhan sistem [VSC-99].

2.6.2 *Data Flow Diagram* (DFD)

DFD adalah suatu teknik yang digunakan untuk proses *modeling*. DFD terdiri dari *data flow*, proses-proses, penyimpanan data, dan entitas eksternal [VSC-99].

2.6.2.1 Komponen DFD

Komponen-komponen dan simbol yang digunakan dalam DFD, yaitu:

a. Komponen Proses

Proses menunjukkan transformasi dari masukan menjadi keluaran dan disimbolkan dengan lingkaran atau segi empat tumpul. Komponen proses umumnya didefinisikan dengan kata kerja sederhana.

b. Komponen Aliran Data (*Data Flow*)

Menggambarkan aliran data atau informasi dari satu bagian ke bagian lain dari sistem, dan menggunakan kata benda untuk mengindikasikan data. Aliran data direpresentasikan dengan menggunakan anak panah. Ujung panah menunjukkan arah data bergerak.

c. Komponen Penyimpanan (*Data Store*)

Digunakan untuk mempresentasikan *logical file*, database, yang dalam sebuah sistem tetap berada di dalam *scope* sistem tersebut. Komponen penyimpanan ini direpresentasikan dengan garis paralel atau segi empat terbuka.

d. Komponen Entitas Eksternal (*Source/Sink*)

Merupakan bagian luar sistem, aliran data yang dihubungkan dengan entitas eksternal dan menunjukkan hubungan antara sistem dengan dunia luar. Entitas ekstenal direpresentasikan dengan empat persegi panjang.

2.6.2.2 DFD *Leveled*

Dalam DFD *leveled* ini akan terjadi penurunan *level* dimana dalam penurunan *level* yang lebih rendah harus mampu merepresentasikan proses tersebut ke dalam spesifikasi proses yang jelas. Dalam penurunan *level* ini ada beberapa aturan yang harus dijalankan, yaitu:

a. Setiap penurunan hanya dilakukan bila perlu.

b. Tidak semua bagian dari sistem harus diturunkan dengan jumlah *level* yang sama.

c. Aliran data yang masuk dan keluar pada suatu proses di *level* X harus berhubungan dengan aliran data yang masuk dan keluar pada *level* X+1 yang mendefinisikan proses pada *level* X tersebut.

BAB III
METODE PENELITIAN

Untuk merealisasikan tujuan penelitian tugas akhir ini, langkah-langkah yang dilakukan adalah sebagai berikut:

3.1 Studi Literatur

Studi literatur yang dilakukan bertujuan untuk mengkaji hal-hal yang berhubungan dengan desain dan implementasi sistem yaitu:

1. Kajian pustaka mengenai *cryptography* khususnya *symmetric key cryptography*.
2. Kajian pustaka mengenai *stream cipher cryptography*.
3. Kajian pustaka mengenai algoritma dan arsitektur Grain.
4. Kajian pustaka untuk analisa matematis algoritma Grain, yaitu: *group theory, ring, finite fields* dan *polynomial.*
5. Kajian pustaka mengenai *Linear Feedback Shift Register* (LFSR) dan *Non linear Feedback Shift Register* (NFSR).
6. Kajian pustaka mengenai *randomness testing.*
7. Kajian pustaka mengenai VHDL sebagai bahasa pemrograman untuk desain IC (*Integrated Circuit*).

3.2 Perancangan

Desain dan implementasi *cryptography processor* ini menggunakan algoritma Grain, yang merupakan salah satu *algoritma stream cipher cryptography*. Pengimplementasian *cryptography processor* ini menggunakan bahasa pemrograman VHDL dengan memanfaatkan Xilinx *ISE WebPACK 8.1i*. Desain *stream cipher cryptography processor* algoritma Grain ini secara umum terdiri dari 12 pin utama dan 2 pin untuk catu daya, yaitu pin `vcc` dan pin `ground`. Pin-pin utama Grain *processor* ini dapat dilihat pada Tabel 3.1.

Tabel 3.1 Konfigurasi Pin Grain *Processor*

No.	Pin	I/O	Fungsi
1.	`clock`	input	Pin untuk clock
2.	`reset`	input	Pin untuk reset
3.	`g0`	input	Pin untuk memilih versi Grain
4.	`g1`	input	Pin untuk memilih versi Grain
5.	`key`	input	Pin untuk memasukkan *key*
6.	`iv`	input	Pin untuk memasukkan *initialization vector*
7.	`pt`	input	Pin untuk memasukkan *plaintext*
8.	$f/\bar{r}$	input	Pin untuk mengaktifkan proses mengisi *key* dan *initialization vector*
9.	`inite`	input	Pin untuk mengakhiri proses inisialisasi
10.	`ee`	input	Pin untuk mengaktikan *keystream*
11.	`ct`	output	Pin untuk *ciphertext*
12.	`keystream`	output	Pin untuk *keystream* yang dihasilkan

Sumber : Perancangan

3.3 Implementasi

Gambaran umum *design cycle* dalam pengimplementasian *stream cipher cryptography processor* algoritma Grain ini adalah:

1. Pengkajian algoritma dan arsitektur Grain.
2. Implementasi Grain dengan menggunakan bahasa pemrograman VHDL.
3. *Logic design tool* yang digunakan adalah Xilinx *ISE WebPACK 8.1i*.
4. *Synthesis Tool* yang digunakan adalah *XST*.
5. Simulator yang digunakan adalah *ISE Simulator*.

3.4 Pengujian dan Analisis

Dalam tugas akhir ini dilakukan dua macam pengujian terhadap *stream cipher cryptograhy processor* algoritma Grain yaitu:

3.4.1 Pengujian Validitas *Keystream*.

Metode pengujian ini dilakukan dengan menggunakan program *testbench*, menggunakan *test vector* Grain.

3.4.2 Pengujian Tingkat Keacakan (*Randomness Testing*)

Pengujian tingkat keacakan dilakukan untuk menguji apakah bit *sequences* dari hasil implementasi telah memenuhi syarat acak. Dalam penelitian tugas akhir ini digunakan metode pengujian tingkat keacakan (*randomness testing*) berdasarkan standar test dari *National Institute of Standard and Technology* [NIS-01] yaitu :

1. *Frequency Test*
2. *Frequency Test within a Block*
3. *Runs Test*
4. *Test for the Longest Run of Ones in a Block*
5. *The Binary Matrix Rank Test*

3.5 Penutup

Dalam bab ini dilakukan pengambilan kesimpulan dari hasil implementasi dan pengujian yang dibuat. Pengambilan kesimpulan ini didasarkan pada kesesuaian antara teori dan praktek.

Tahap terakhir dari penelitian ini adalah penulisan saran yang ditujukan untuk memperbaiki kesalahan-kesalahan yang terjadi serta menyempurnakan penelitian.

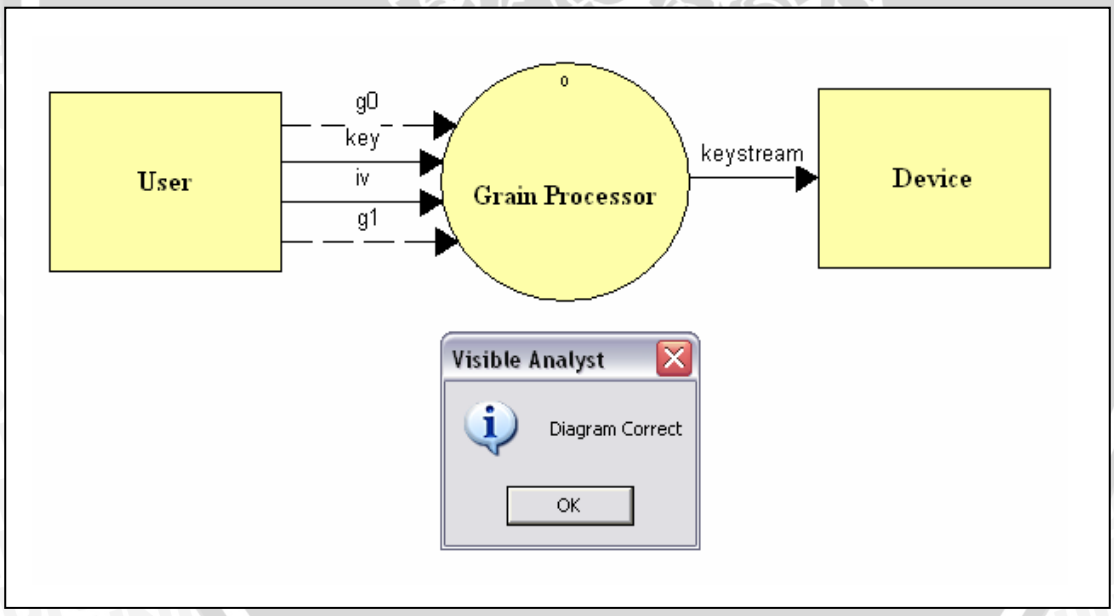
BAB IV
PERANCANGAN

4.1 Deskripsi Implementasi

Pada desain dan implementasi *stream cipher crptography processor* algoritma Grain ini dibuat sebuah *cryptography processor* dengan menggunakan algoritma *stream cipher* yaitu Grain. *Stream cipher crptography processor* algoritma Grain ini dapat digunakan untuk aplikasi yang membutuhkan keamanan dalam transfer data.

4.2 Diagram Konteks

Diagram konteks merupakan sebuah diagram sederhana yang menggambarkan hubungan dengan entitas luar, masukan dan keluaran sistem. Diagram konteks Grain *stream cipher cryptography processor* ditunjukkan dalam Gambar 4.1.



Gambar 4.1 Diagram Konteks Grain *Processor*
Sumber : Perancangan

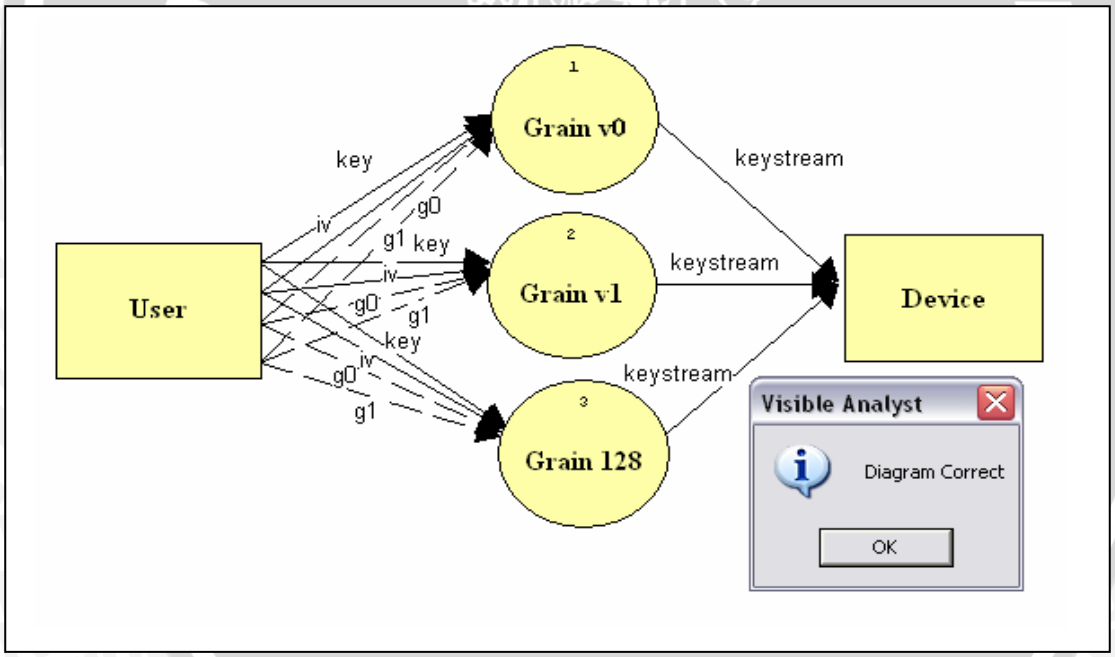
Diagram konteks mengambarkan proses yang terjadi secara umum antara pengguna atau *user* yang akan memberikan masukan berupa kunci atau *key* dan *initialization vector* (IV), serta kontrol g0 dan g1 untuk memilih algoritma Grain yang akan digunakan. Hasil dari Grain processor yang berupa *keystream* akan digunakan

oleh *device* lain yang membutuhkan keamanan data, misalnya *mobile phone* untuk GSM *System*.

4.3 *Data Flow Diagram* (DFD)

DFD menggambarkan penyimpanan data dan proses yang mentransformasikan data. DFD menunjukkan hubungan antara data pada sistem dan proses pada sistem. DFD ini dirancang menggunakan perangkat lunak Visible Analyst 7.4 dengan aturan Yourdon /DeMarco.

Diagram konteks mengambarkan proses ayng terjadi secara umum. Diagram konteks dari *stream cipher cryptography processor* ditunjukkan dalam Gambar 4.1. Proses di dalam diagram konteks dapat dijabarkan menggunakan *data flow diagam*. *Data flow diagram level 0* untuk *stream cipher cryptography processor* algoritma Grain ditunjukkan dalam Gambar 4.2.



Gambar 4.2 *Data Flow Diagram level 0*
Sumber : Perancangan

Data flow diagram level 0 untuk *stream cipher cryptography processor* algoritma Grain terdiri atas tiga macam proses, yaitu :

1. Proses 1 – Grain v0

   Proses Grain v0 ini memiliki masukan berupa *key* dan *initialization vector* yang akan diolah untuk menghasilkan *keystream* berdasarkan algoritma Grain v0.

2. Proses 2 – Grain v1

   Pada proses Grain v1 ini, data masukan yang berupa *key* dan *initialization vector* ini akan diolah berdasarkan algoritma Grain v1 untuk menghasilkan *keystream*.

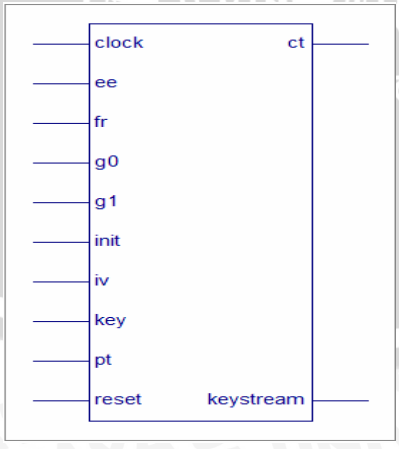
3. Proses 3 – Grain 128

   Proses ketiga pada *data flow diagram level 0* ini adalah proses Grain 128. Pada proses ini masukan yang berupa *key* dan *initialization vector* akan diproses berdasarkan algoritma Grain 128. Hasil dari proses ini berupa *keystream*.

Ketiga proses pada *data flow diagram level 0* untuk *stream cipher cryptography processor* algoritma Grain tersebut dapat dipilih melalui dua buah kontrol yang digunakan untuk menentukan pilihan algoritma Grain. Kedua macam kontrol tersebut adalah g0 dan g1 yang juga merupakan masukan pada proses Grain v0, Grain v1 dan Grain 128. Kombinasi kedua kontrol tersebut digunakan untuk menentukan pilihan algoritma yang digunakan.

4.4 Desain *Cryptography Processor*

Desain *stream cipher crptography processor* algoritma Grain ini terdiri dari 12 pin utama dan 2 pin untuk catu daya, yaitu pin `vcc` dan pin `ground`. Pin-pin utama *processor* ini dapat dilihat pada Tabel 4.1 dan dalam Gambar 4.1.



Gambar 4.1 Desain Grain *Processor*

Tabel 4.1 Konfigurasi Pin Grain *Processor*

No.	Pin	I/O	Fungsi
1.	`Clock`	Input	Pin untuk *clock*
2.	`Reset`	Input	Pin untuk *reset*
3.	`g0`	Input	Pin untuk memilih versi Grain
4.	`g1`	Input	Pin untuk memilih versi Grain
5.	`Key`	Input	Pin untuk memasukkan *key*
6.	`Iv`	Input	Pin untuk memasukkan *initialization vector*
7.	`Pt`	Input	Pin untuk memasukkan *plaintext*
8.	$f/\bar{r}$	Input	Pin untuk mengaktifkan proses mengisi *key* dan *initialization vector*
9.	`Inite`	Input	Pin untuk mengakhiri proses inisialisasi
10.	`Ee`	Input	Pin untuk mengaktikan *keystream*
11.	`Ct`	output	Pin untuk *ciphertext*
12.	`Keystream`	output	Pin untuk *keystream* yang dihasilkan

Sumber : Perancangan

Tahap pertama yang dilakukan dalam tahap perancangan adalah mendesain komponen utama, yaitu komponen `Grain_Processor`. Komponen ini digunakan untuk memilih versi Grain dengan menggunakan pin *`g0`* dan *`g1`*. Kombinasi pin tersebut dapat dilihat pada Tabel 4.2.

Tabel 4.2 Kombinasi Pin `g0` dan `g1`

g1	g0	Versi Grain
0	0	*Not used*
0	1	Grain v0
1	0	Grain v1
1	1	Grain 128

Sumber : Perancangan

```
IF g0=1 and g1=0
      keystream  := keystream Grainv0
      ciphertext := ciphertext Grainv0
ELSE IF g0=0 and g1=1
      keystream  := keystream Grainv1
      ciphertext := ciphertext Grainv1
ELSE IF g0=1 and g1=1
      keystream  := keystream Grain128
      ciphertext := ciphertext Grain128
ELSE
      keystream  := 0
      ciphertext := 0
END IF
```

Pada komponen utama ini, pin *key* dan *iv* masing-masing digunakan untuk masukan kunci (*key*) dan *initialization vector (iv)* dari Grain *stream cipher crptography processor*. Kedua pin ini akan aktif selama pin $f/\bar{r}$ berlogika 1, dengan banyak *clock* tergantung pada versi Grain yang dipilih.

Saat $f/\bar{r}$ berlogika 0, proses inisialisasi dimulai, dan akan berakhir setelah pin *inite* berlogika 1. *Keystream* akan dihasilkan saat pin *ee* aktif (berlogika 1). Data yang akan dienkripsi akan menjadi masukan bagi pin *pt*, dan mulai dienkripsi saat *keystream* mulai dihasilkan dari pin *keystream*. Pin *keystream* ini digunakan untuk melihat hasil *keystream*. Sedangkan data hasil enkripsi akan dihasilkan melalui pin *ct*.

Komponen utama ini terdiri dari 3 komponen algoritma Grain yang berbeda, yaitu komponen `Grainv0`, `Grainv1`, dan `Grain128`.

4.4.1 Modul Grain v0

Pada komponen `Grainv0` ini, panjang kunci dan panjang *initialization vctor* yang digunakan masing-masing adalah 80 bit. Desain pada komponen Grainv0 ini terdiri dari 10 pin, seperti pada Tabel 4.3.

Tabel 4.3 Konfigurasi Pin Komponen `Grainv0`

No.	Pin	I/O	Fungsi
1.	`clock`	input	Pin untuk *clock*
2.	`reset`	input	Pin untuk *reset*
3.	`key`	input	Pin untuk memasukkan *key*
4.	`iv`	input	Pin untuk memasukkan *initialization vector*
5.	`pt`	input	Pin untuk memasukkan *plaintext*
6.	$f/\bar{r}$	input	Pin untuk mengaktifkan proses mengisi *key* dan *initialization vector*
7.	`inite`	input	Pin untuk mengakhiri proses inisialisasi
8.	`ee`	input	Pin untuk mengaktikan *keystream*
9.	`ct`	output	Pin untuk *ciphertext*
10.	`keystream`	output	Pin untuk *keystream* yang dihasilkan

Sumber : Perancangan

Algoritma program Grain *stream cipher crptography processor* ini berdasarkan arsitektur Grain yang telah dibahas pada Bab II, yang terdiri dari 3 bagian utama. Pada komponen `Grainv0` ini terdiri dari 3 bagian utama, yaitu *linear feedback shift register, non linear feedback shift register* dan *non linear filter.*

4.4.1.1 *Linear Feedback Shift Register*

Pada bagian program *linear feedback shift register* dibuat dengan menggunakan komponen `d_register` yang dihubungkan dengan *signal* `QL1` sampai `QL80`. Nilai kembalian untuk *linear feedback shift register* ini berdasarkan fungsi f(x), kemudian hasilnya akan dihubungkan ke *signal* `res_lfsr`.

```
res_lfsr  := QL18 xor QL29 xor QL42 xor QL57 xor QL67 xor QL80
```

Saat proses inisialisasi kunci, *signal* `res_lfsr` dan hasil keluaran dari Grain *cipher* akan membentuk fungsi kembalian, yaitu pada saat pin `inite` tidak diaktifkan. Saat proses inisialisasi kunci tersebut, *signal* `res_lfsr` akan dihubungkan dengan *signal* `feedback_lfsr`. *Signal* `feedback_lfsr` ini akan menjadi masukan bagi

linear feedback shift register jika pin $f/\bar{r}$ berlogika 0. Jika pin $f/\bar{r}$ berlogika 1, maka masukan pin *iv* akan menjadi masukan bagi *signal* `feedback_lfsr`.

```
IF f/r̄ =1
    feedback_lfsr := iv
ELSE
    result := not inite and output
    feedback_lfsr := res_lfsr xor result
END IF
```

4.4.1.2 *Non Linear Feedback Shift Register*

Komponen `d_register` juga dimanfaatkan untuk bagian program *non linear feedback shift register*. Komponen `d_register` ini dihubungkan dengan *signal* `QN1` sampai `QN80`. Nilai kembalian untuk *non linear feedback shift register* ini berdasarkan fungsi g(x) yang hasilnya dihubungkan dengan *signal* `res_nfsr`. Untuk desain bagian ini terdapat beberapa *signal* tambahan, yaitu *signal* `temp0` sampai `temp11`. Masing-masing *signal* digunakan untuk membentuk fungsi *non linear* g(x).

```
res_nfsr :=  temp11 xor temp0 xor temp1 xor temp2 xor temp3 xor
             temp4 xor temp5 xor temp6 xor temp7 xor temp8 xor
             temp9 xor temp10
```

Signal `res_nfsr` dan hasil keluaran dari Grain *cipher* akan membentuk fungsi kembalian saat pin *inite* tidak diaktifkan, yaitu pada saat proses inisialisasi kunci. *Signal* `res_nfsr` tersebut akan dihubungkan dengan *signal* `feedback_nfsr`. *Signal* `feedback_nfsr` ini akan menjadi masukan bagi *non linear feedback shift register* jika pin $f/\bar{r}$ berlogika 0. Jika pin $f/\bar{r}$ berlogika 1, maka masukan pin *key* akan menjadi masukan bagi *signal* `feedback_nfsr`.

```
IF  f / r̄ =1
    feedback_nfsr := key
ELSE
    result := not inite and output
    feedback_nfsr := res_nfsr xor result
END IF
```

4.4.1.3 *Non Linear Filter*

Pada *non linear filter* ini dilakukan *filtering* dengan menggunakan fungsi h(x) yang merupakan fungsi *non linear* dengan variabel yang diperoleh dari *linear feedback shift register* dan *non linear feedback shift register*. Untuk membentuk fungsi ini pada program VHDL dibutuhkan beberapa *signal*, yaitu *signal* `filter0` sampai `filter8`. Masing-masing *signal* digunakan untuk membentuk fungsi h(x). Hasil dari fungsi ini akan dihubungkan dengan *signal* `output`.

```
nlf     :=  filter0 xor filter1 xor filter2 xor filter3 xor filter4
            xor filter5 xor filter6 xor filter7 xor filter8
output  :=  QN80 xor nlf
```

Desain pada bagian ini merupakan hal yang paling penting karena berhubungan langsung dengan *keystream* yang digunakan untuk mengenkripsi data. *Keystream* ini akan dihasilkan jika pin *ee* diaktifkan.

```
stream      := output and ee
keystream   := stream
```

4.4.2 Modul Grain v1

Komponen `Grainv1` ini memiliki 10 pin, yang terdiri dari 8 pin input dan 2 pin output. Konfigurasi pin untuk komponen `Grainv1` dapat dilihat pada Tabel 4.4.

Tabel 4.4 Konfigurasi Pin Komponen `Grainv1`

No.	Pin	I/O	Fungsi
1.	`Clock`	input	Pin untuk *clock*
2.	`Reset`	input	Pin untuk *reset*
3.	`Key`	input	Pin untuk memasukkan *key*
4.	`Iv`	input	Pin untuk memasukkan *initialization vector*
5.	`Pt`	input	Pin untuk memasukkan *plaintext*
6.	$f/\bar{r}$	input	Pin untuk mengaktifkan proses mengisi *key* dan *initialization vector*
7.	`Inite`	input	Pin untuk mengakhiri proses inisialisasi
8.	`Ee`	input	Pin untuk mengaktikan *keystream*
9.	`Ct`	output	Pin untuk *ciphertext*
10.	`keystream`	output	Pin untuk *keystream* yang dihasilkan

Sumber : Perancangan

Sama halnya dengan komponen `Grainv0`, pada desain program komponen `Grainv1` ini juga terdiri dari 3 bagian utama yaitu *linear feedback shift register, non linear feedback shift register* dan *non linear filter*.

4.4.2.1 *Linear Feedback Shift Register*

Bagian program *linear feedback shift register* pada komponen `Grainv1` ini terdiri dari komponen `d_register` yang dihubungkan dengan *signal* `QL1` sampai `QL80`. Nilai kembalian *linear feedback shift register* ini berdasarkan persamaan f(x), yang hasilnya dihubungkan dengan *signal* `res_lfsr`.

```
res_lfsr  := QL18 xor QL29 xor QL42 xor QL57 xor QL67 xor QL80
```

Saat proses inisialisasi kunci, *signal* `res_lfsr` dan hasil keluaran dari Grain *cipher* akan membentuk fungsi kembalian saat pin `inite` tidak diaktifkan, dan s*ignal* `res_lfsr` tersebut akan dihubungkan dengan *signal* `feedback_lfsr`. Jika pin $f/\bar{r}$ berlogika 0, *signal* `feedback_lfsr` ini akan menjadi masukan bagi *linear feedback*

shift register. Jika pin $f/\bar{r}$ berlogika 1, maka masukan pin *iv* akan menjadi masukan bagi *signal* feedback_lfsr.

```
IF f/r̄ =1
    feedback_lfsr := iv
ELSE
    result := not inite and output
    feedback_lfsr := res_lfsr xor result
END IF
```

4.4.2.2 *Non Linear Feedback Shift Register*

Non linear feedback shift register merupakan bagian program yang memiliki nilai kembalian berupa fungsi g(x) yang merupakan fungsi *non linear*. Komponen d_register digunakan untuk membentuk *non linear feedback shift register* dengan cara menghubungkan *signal* QN1 sampai QN80. Berbeda dengan algoritma Grain v0, pada algoritma Grain v1 terdapat beberapa nilai kembalian yang diambil dari variabel yang berbeda. Desain fungsi g(x) ini membutuhkan beberapa tambahan untuk membentuk fungsi g(x), yaitu *signal* temp0 sampai temp11 yang akan hasilnya dihubungkan dengan *signal* res_nfsr.

```
res_nfsr  :=  temp11 xor temp0 xor temp1 xor temp2 xor temp3 xor
              temp4 xor temp5 xor temp6 xor temp7 xor temp8 xor
              temp9 xor temp10
```

Proses inisialisasi kunci dimulai setelah kunci dan *initialization vector* dimasukkan. *Signal* res_nfsr dan hasil keluaran dari Grain *cipher* akan membentuk fungsi kembalian saat pin *inite* tidak aktif, yaitu pada saat proses inisialisasi kunci. *Signal* res_nfsr tersebut akan dihubungkan dengan *signal* feedback_nfsr. Jika pin $f/\bar{r}$ berlogika 0, *signal* feedback_nfsr ini akan menjadi masukan bagi *non linear feedback shift register*. Jika pin $f/\bar{r}$ berlogika 1, maka masukan pin *key* akan menjadi masukan bagi *signal* feedback_nfsr.

```
IF f/r̄ =1
    feedback_nfsr := key
ELSE
    result := not inite and output
    feedback_nfsr := res_nfsr xor result
END IF
```

4.4.2.3 *Non Linear Filter*

Pada *non linear filter* ini dilakukan *filtering* dengan menggunakan fungsi h(x) yang merupakan fungsi *non linear* dengan variabel yang diperoleh dari *linear feedback shift register* dan *non linear feedback shift register*. Untuk membentuk fungsi ini pada program VHDL dibutuhkan beberapa *signal* yaitu signal `filter0` sampai `filter8`, dan dihubungkan dengan *signal* `output`. Pada komponen `Grainv1` ini fungsi h(x) yang digunakan untuk menghasilkan *keystream* dibuat lebih kompleks untuk menghasilkan *keystream* yang lebih kuat, sehingga lebih sulit untuk dipecahkan.

```
nlf    := filter0 xor filter1 xor filter2 xor filter3 xor
          filter4 xor filter5 xor filter6 xor filter7 xor
          filter8
output := QN79 xor QN78 xor QN76 xor QN70 xor QN49
          xor QN37 xor QN24 XOR nlf
```

Non linear filter ini didesain untuk menghasilkan *keystream* yang digunakan untuk mengenkripsi data. *Keystream* ini akan dihasilkan jika pin `ee` aktif.

```
stream    := output and ee
keystream := stream
```

4.4.3 Modul Grain 128

Berbeda dengan Grain versi sebelumnya, pada Grain 128 ini panjang kunci dan *initialization vector* yang digunakan berbeda. Pada Grain 128 ini, panjang kunci dan *initialization vector* yang digunakan adalah 128 bit. Dengan kunci dan

initialization vector yang lebih panjang maka akan meningkatkan kekuatan *keystream* yang dihasilkan, sehingga lebih sulit untuk pecahkan. Tetapi proses inisialisasi kunci pada Grain 128 ini lebih lambat. Konfigurasi pin pada komponen `Grain128` ini dapat dilihat pada Tabel 4.5.

Tabel 4.5 Konfigurasi Pin Komponen `Grain128`

No.	Pin	I/O	Fungsi
1.	`clock`	input	Pin untuk *clock*
2.	`reset`	input	Pin untuk *reset*
3.	`key`	input	Pin untuk memasukkan *key*
4.	`iv`	input	Pin untuk memasukkan *initialization vector*
5.	`pt`	input	Pin untuk memasukkan *plaintext*
6.	$f/\bar{r}$	input	Pin untuk mengaktifkan proses mengisi *key* dan *initialization vector*
7.	`inite`	input	Pin untuk mengakhiri proses inisialisasi
8.	`ee`	input	Pin untuk mengaktikan *keystream*
9.	`ct`	output	Pin untuk *ciphertext*
10.	`keystream`	output	Pin untuk *keystream* yang dihasilkan

Sumber : Perancangan

Secara garis besar, arsitektur Grain 128 ini hampir sama dengan Grain versi sebelumnya, yang terdiri dari *linear feedback shift register, non linear feedback shift register* dan *non linear filter.*

4.4.3.1 *Linear Feedback Shift Register*

Pada bagian program *linear feedback shift register*, *signal* `QL1` sampai `QL128` digunakan untuk menghubungkan komponen `d_register`. Nilai kembalian untuk *linear feedback shift register* ini berdasarkan fungsi f(x) pada Grain 128. Hasil dari fungsi ini dihubungkan dengan *signal* `res_lfsr`. Signal ini akan dihubungkan dengan *signal* `feedback_lfsr.`

```
res_lfsr := QL32 xor QL47 xor QL58 xor QL90 xor QL121 xor QL128
```

Setelah *linear feedback shift register* dan *non linear feedback shift register* maing-masing telah terisi dengan *initialization vector* dan kunci, proses inisialisasi kunci dimulai. *Signal* `res_lfsr` dan hasil keluaran dari Grain *cipher* akan membentuk fungsi kembalian saat pin `inite` tidak aktif. *Signal* `res_lfsr` tersebut akan dihubungkan dengan *signal* `feedback_lfsr`. *Signal* `feedback_lfsr` ini akan menjadi masukan bagi *linear feedback shift register* jika pin $f/\bar{r}$ berlogika 0. Jika pin $f/\bar{r}$ berlogika 1, maka masukan pin `iv` akan menjadi masukan bagi *signal* `feedback_lfsr`.

```
IF f/r̄ =1
    feedback_lfsr := iv
ELSE
    result := not inite and output
    feedback_lfsr := res_lfsr xor result
END IF
```

4.4.3.2 *Non Linear Feedback Shift Register*

Non linear feedback shift register merupakan bagian program yang memiliki nilai kembalian berupa fungsi g(x) yang merupakan fungsi *non linear*. Bagian ini menggunakan komponen `d_register` yang dihubungkan dengan *signal* `QN1` sampai `QN128`. Fungsi kembalian *non linear feedback shift register* ini berdasarkan persamaan g(x). Desain fungsi g(x) ini membutuhkan beberapa *signal* yang akan menghasilkan `res_nfsr`, yaitu *signal* `temp0` sampai `temp6` dan `temp11`.

```
res_nfsr :=  temp11 xor temp0 xor temp1 xor temp2 xor temp3
             xor temp4 xor temp5 xor temp6
```

Signal `res_nfsr` dan hasil keluaran dari Grain *cipher* akan membentuk fungsi kembalian saat pin `inite` tidak diaktifkan, yaitu pada saat proses inisialisasi kunci. *Signal* `res_nfsr` tersebut akan dihubungkan dengan *signal* `feedback_nfsr`.

Signal `feedback_nfsr` ini akan menjadi masukan bagi *non linear feedback shift register* jika pin $f/\bar{r}$ berlogika 0. Jika pin $f/\bar{r}$ berlogika 1, maka masukan pin *key* akan menjadi masukan bagi *signal* `feedback_nfsr`.

```
IF  f/r̄ =1
    feedback_nfsr := key
ELSE
    result := not inite and output
    feedback_nfsr := res_nfsr xor result
END IF
```

4.4.3.3 *Non Linear Filter*

Pada komponen `Grain128` ini fungsi h(x) yang digunakan untuk menghasilkan *keystream* dibuat lebih kompleks untuk menghasilkan *keystream* yang lebih kuat, yang lebih sulit untuk pecahkan. Fungsi h(x) merupakan fungsi *non linear* dengan variabel yang diperoleh dari *linear feedback shift register* dan *non linear feedback shift register*.

Untuk membentuk fungsi ini pada program VHDL dibutuhkan beberapa *signal* yang digunakan untuk menghasilkan fungsi h(x). Signal *`filter0`* sampai *`filter4`* digunakan untuk membantu membentuk persaman tersebut, dan hasilnya akan dihubungkan ke *signal* `output`.

```
nlf     := filter0 xor filter1 xor filter2 xor filter3 xor
           filter4
output  := QN126 xor QN113 xor QN92 xor QN83 xor QN64 xor
           QN55 xor QN39 xor QL35 xor nlf
```

Non linear filter ini didesain untuk menghasilkan *keystream* yang digunakan untuk mengenkripsi data. *Keystream* ini akan dihasilkan jika pin *ee* aktif.

```
stream      := output and ee
keystream   := stream
```

BAB V
IMPLEMENTASI

5.1 Lingkungan Implementasi

Desain dan implementasi *stream cipher cryptography processor* algoritma Grain ini dilakukan dalam lingkungan :

Perangkat lunak yang digunakan dalam implementasi ini adalah :

1. Sistem operasi yang digunakan adalah *Microsoft Windows XP Profesional*.
2. Bahasa pemrograman yang digunakan adalah VHDL.
3. *Logic design tool* yang digunakan adalah Xilinx *ISE WebPACK 8.1i.*
4. *Synthesis Tool* yang digunakan adalah *XST*.
5. *Simulator* yang digunakan adalah *ISE Simulator*.

Perangkat keras yang digunakan dalam implementasi ini adalah :

1. Komputer Pentium IV CPU 2.66GHz
2. Memory 192 MB RAM.

5.2 Implementasi *Cryptography Processor*

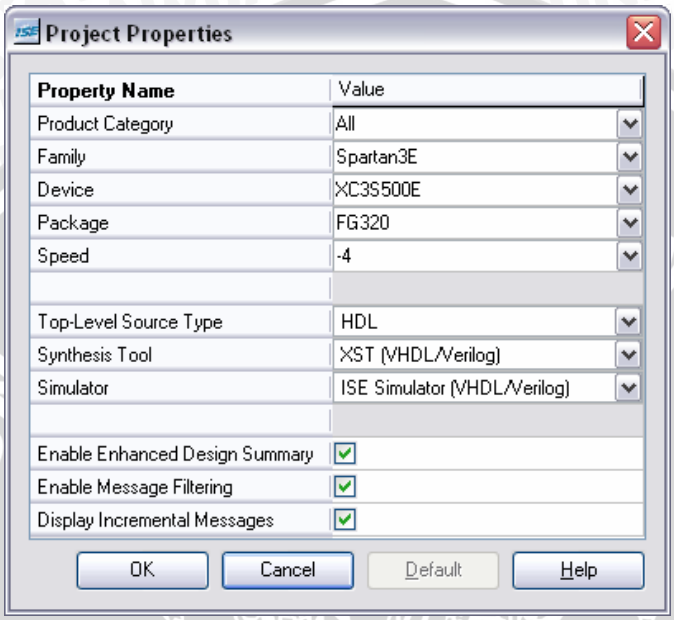
Dari perancangan yang telah dibuat dalam Bab IV, desain dan implementasi *stream cipher cryptography processor* algoritma Grain ini terdiri dari beberapa langkah yang akan dibahas lebih lanjut pada bagian berikut.

5.2.1 Tahap Spesifikasi

Dalam desain dan implementasi *stream cipher cryptography processor* algoritma Grain ini *logic design tool* yang digunakan adalah Xilinx *ISE WebPACK 8.1i*. Langkah pertama adalah dengan menentukan spesifikasi *device* yang akan digunakan. *Logic device* Xilinx yang dapat digunakan dapat berupa CPLD atau FPGA dengan berbagai jenis *family* dan tipe *device*. Untuk pembuatan *stream cipher cryptography processor* algoritma Grain ini, digunakan *FPGA* Xilinx *Spartan3E XC3S500 E* dengan *package FG320*, seperti terlihat dalam Gambar 5.1.

Tahap berikutnya adalah mendeskripsikan *behavioral* dari *stream cipher cryptography processor* algoritma Grain yang akan dibuat. Berdasarkan hasil perancangan sebelumnya, *stream cipher cryptography processor* algoritma Grain ini terdiri dari beberapa komponen yang akan digunakan. Komponen yang paling penting

adalah `d_register` yang merupakan komponen dari `Grainv0`, `Grainv1` dan `Grain128`. Pendeskripsian *behavioral* modul `Grainv0`, `Grainv1` dan `Grain128`. didasarkan pada algoritma Grain yang telah dibahas pada Bab II. Desain program secara garis besar telah dibahas pada Bab IV, dan *listing* program secara lengkap dapat dilihat pada Lampiran 1.



Gambar 5.1 *Project Device Properties*

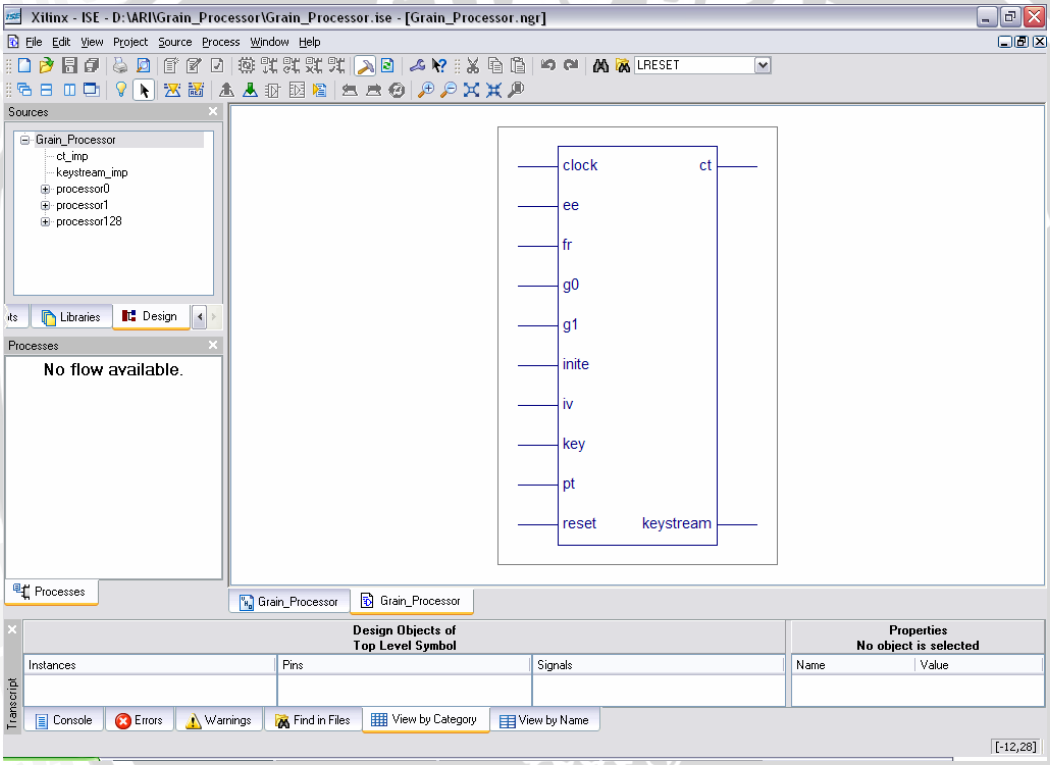
Setelah seluruh proses pendeskripsian selesai, langkah berikutnya akan dibahas lebih lanjut pada bagian berikut.

5.3.2 Tahap Sintesis

Pada tahap sintesis ini, hasil dari desain *behavioral* yang telah dilakukan akan dikonversi menjadi gerbang-gerbang logika dasar yang akan menghasilkan *netlist* yang sesuai dengan spesifikasi *device- family* yang digunakan.

Pada Xilinx *ISE WebPACK 8.1i*, untuk tahap ini disediakan menu *Synthesis/Implementation* pada *Source Window* dan *Synthesis – XST* pada *Process Window*. Langkah pertama yang harus dilakukan adalah dengan menggunakan fasilitas *check syntax* untuk mengecek kesalahan penulisan program VHDL sebelum disintesis. Jika pengecekan sintaks ini berhasil, maka akan muncul pesan pada *console tab* yang menyatakan bahwa tidak ada kesalahan sintaks.

Pada tahap berikutnya, *file* .vhd yang telah dibuat telah siap disintesis dengan menggunakan fasilitas *Synthesis – XST*. Jika tahap sintesis ini berhasil, maka pada *console tab* akan muncul pesan bahwa proses sintesis ini telah berhasil dan akan menghasilkan *Synthesis Report*. *Synthesis Report* pada pembuatan *stream cipher cryptography processor* algoritma Grain ini meliputi hasil Simulasi dan Analisis HDL, *timing report* serta gerbang-gerbang logika yang digunakan. *Synthesis Report* selengkapnya dapat dilihat pada Lampiran 2. Pada tahap ini juga telah dapat dihasilkan RTL diagram dari *stream cipher cryptography processor* algoritma Grain yang dapat dilihat dalam Gambar 5.2.

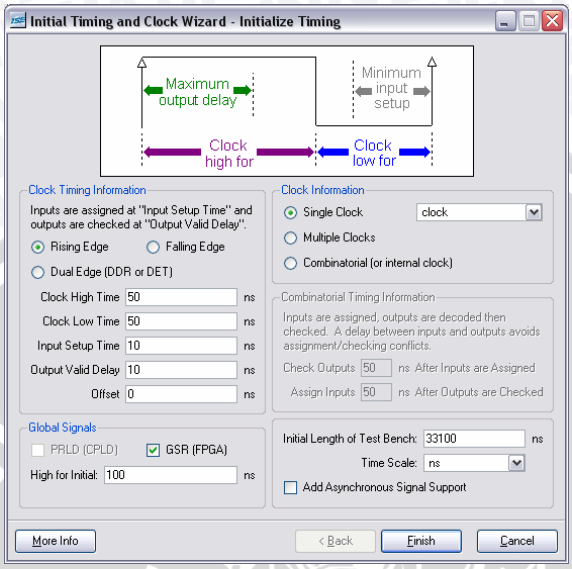


Gambar 5.2. *RTL Schematic*

5.2.3 Tahap Simulasi

Pada tahap simulasi ini digunakan *ISE Simulator* untuk mengecek hasil desain dengan menggunakan *file Test Bench WaveForm*. Pada *file* ini harus ditentukan *timing* yang akan digunakan untuk proses simulasi, yang meliputi periode, input *set-up time* dan *valid-output* serta panjang *test bench* yang digunakan. Pada simulasi ini *Clock High Time* dan *Clock Low Time* diset sebesar 50 ns, sehingga setiap periode *clock* adalah 100 ns. Untuk pengimplementasian *stream cipher cryptography processor* algoritma Grain ini *initial timing set up* dapat dilihat dalam Gambar 5.3.

File Test Bench WaveForm digunakan untuk mensimulasikan *behavioral stream cipher cryptography processor* algoritma Grain dengan memberikan input-input yang diinginkan sesuai desain.



Gambar 5.3 *Initial Timing and Clock*

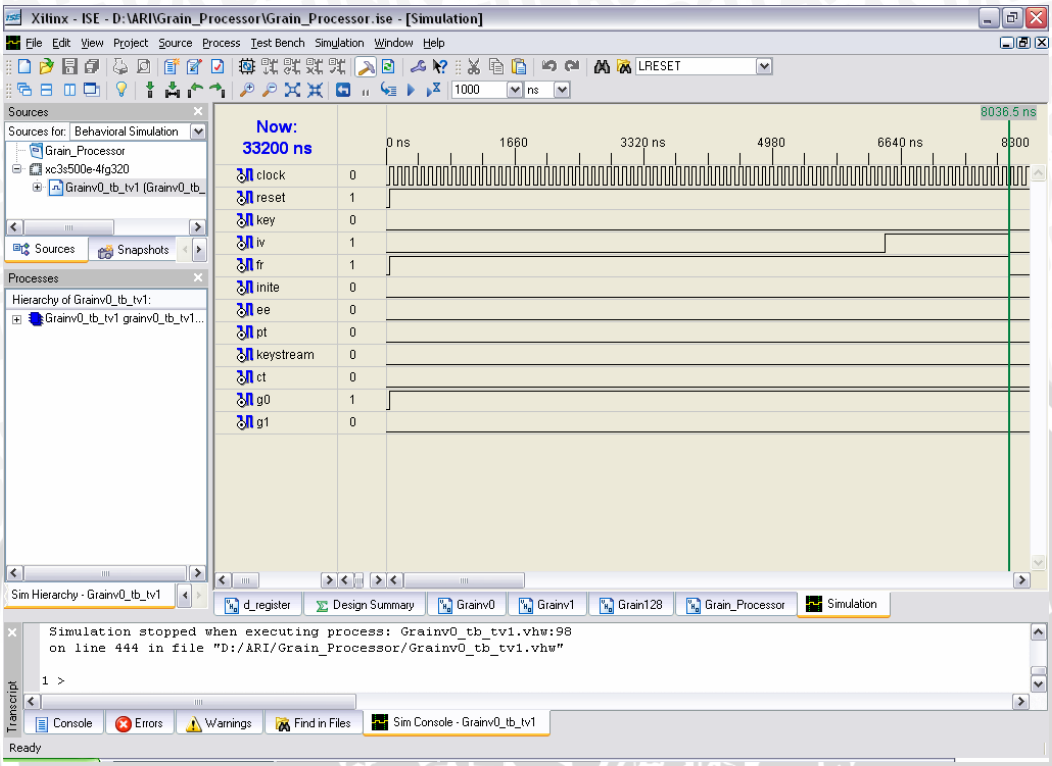
Fasilitas untuk simulasi ini adalah *Behavioral Simulation* pada *Source Window* dan *Simulate Behavoiural Model* pada *Process Window*. Dari simulasi tersebut dapat dilihat apakah *output* yang dihasilkan telah sesuai dengan *output* yang diinginkan.

Hasil simulasi pengimplementasian *stream cipher cryptography processor* algoritma Grain ini dapat dilihat dalam Gambar 5.4 sampai Gambar 5.9. Pada simulasi ini, *Clock High Time* dan *Clock Low Time* diset 50 ns, sehingga setiap periode *clock* adalah 100 ns. C*lock* aktif pada saat *rising edge* dan pin *reset* berlogika 1.

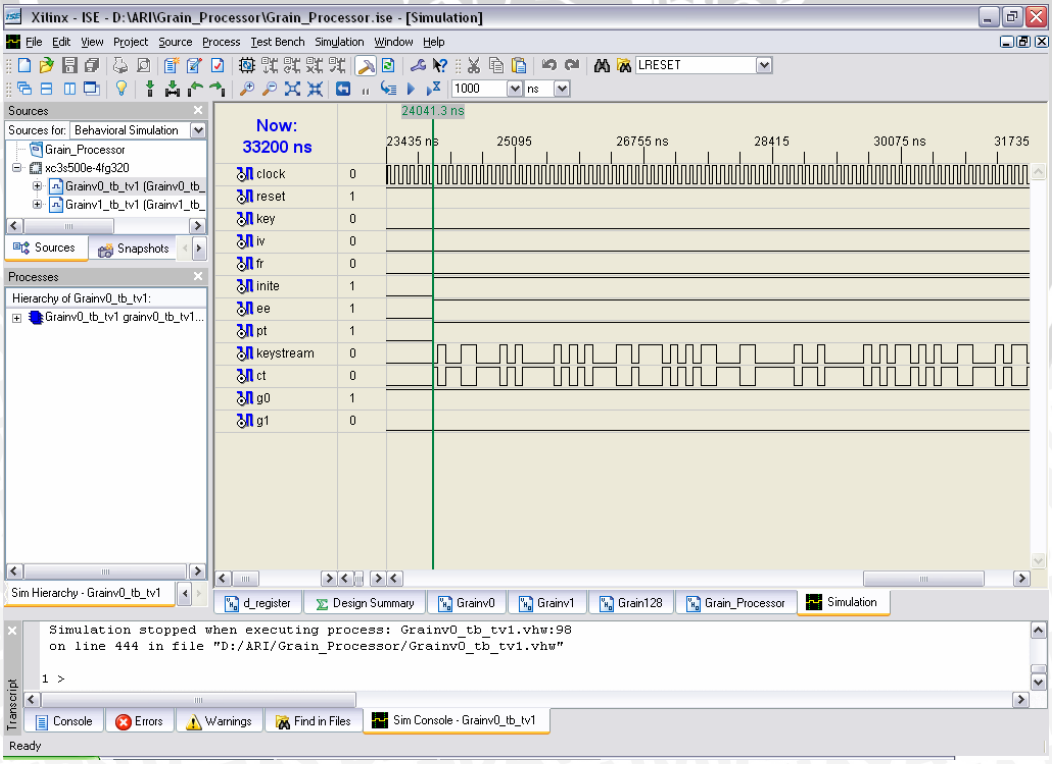
Saat pin `g0` berlogika 1 dan pin `g1` berlogika 0, maka versi Grain yang dipilih adalah Grain v0. Kunci sebanyak 80 bit yang akan digunakan, pertama kali harus di-*load* melalui pin *key* sebanyak 80 *clock*. Pada saat yang sama, 64 bit *initialization vector*, di-*load* pada pin `iv`, dengan 16 bit sisa diisi dengan logika 1, dengan memberikan logika 1 pada pin $f/\bar{r}$ seperti dalam Gambar 5.4.

Sebanyak 160 *clock* berikutnya, dilakukan proses inisialisasi kunci, dengan mengaktifkan pin `inite` pada *clock* ke 240 untuk mengakhiri proses inisialisasi. Untuk melihat *keystream* yang dihasilkan, pin `ee` diaktifkan saat *clock* ke 240, seperti dalam Gambar 5.5. Data yang akan dienkripsi di-*load* pada *clock* ke 240 melalui pin

pt. Dengan *key* dan *iv* yang diberikan, hasil *keystream* tersebut telah sesuai dengan *test vector* yang telah diberikan oleh Grain v0.



Gambar 5.4 Hasil Simulasi Grain v0 Saat Pin f / $\bar{r}$ Berlogika 0 Saat *Clock* ke 80

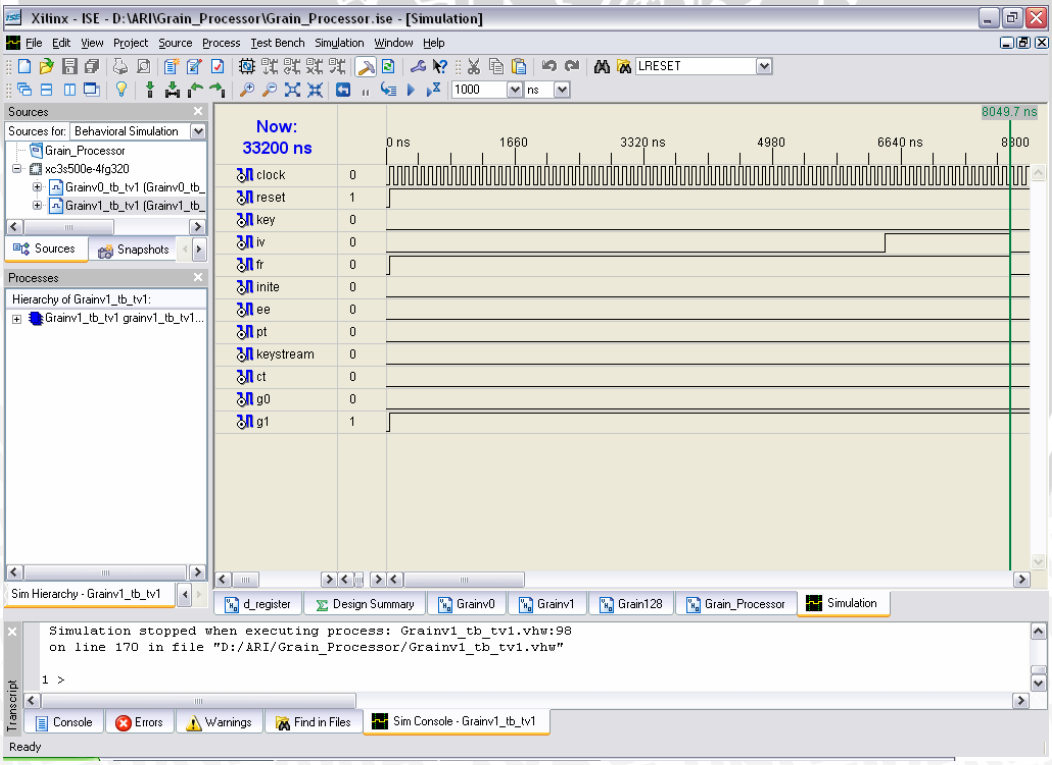


5.5 Hasil Simulasi Grain v0 Saat Pin *inite* dan *ee* Aktif Saat *Clock* ke 240

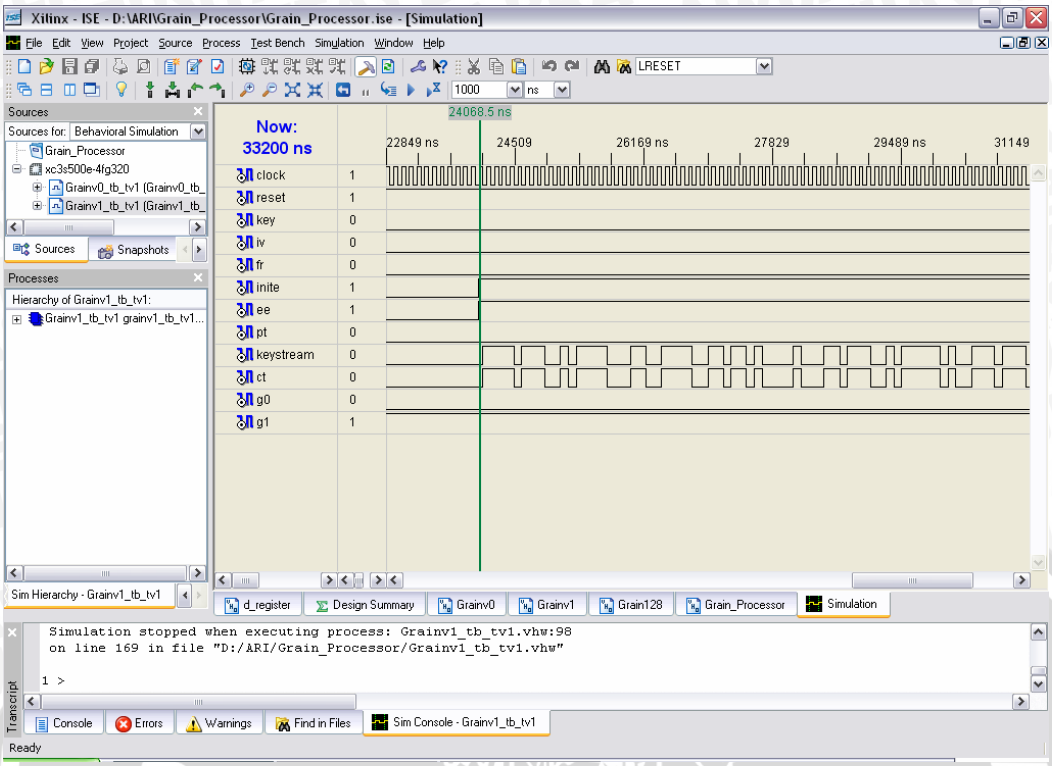
Gambar 5.6 sampai Gambar 5.7 menunjukkan hasil simulasi Grain v1. Untuk mengaktifkan Grain v1, pin `g0` diset berlogika 0 dan pin `g1` diset berlogika 1. Kunci sebanyak 80 bit yang akan digunakan, pertama kali harus di-*load* melalui pin `key` sebanyak 80 *clock*, dengan memberikan logika 1 pada pin $f/\bar{r}$ pada 80 bit pertama. Pada saat yang sama, 64 bit *initialization vector*, di-*load* pada pin `iv` sebanyak 80 *clock*, dengan 16 bit sisa diisi dengan logika 1, seperti dalam Gambar 5.6. Proses tersebut sama dengan proses pada Grain v0.

Saat pin $f/\bar{r}$ berlogika 0, proses inisialisasi kunci dilakukan dan akan berakhir saat pin `inite` diaktifkan pada *clock* ke 240. *Keystream* dapat dihasilkan dengan mengaktifkan pin `ee` saat *clock* ke 240, seperti dalam Gambar 5.7. Untuk proses enkripsi, data di-*load* pada *clock* ke 240 melalui pin `pt`. *Ciphertext* yang dihasilkan dapat dilihat melaui pin `ct`.

Dengan *key* dan *iv* yang diberikan, hasil *keystream* tersebut telah sesuai dengan *test vector* yang telah diberikan oleh Grain v1.



Gambar 5.6 Hasil Simulasi Grain v1 Saat Pin $f/\bar{r}$ Berlogika 0 Saat *Clock* ke 80

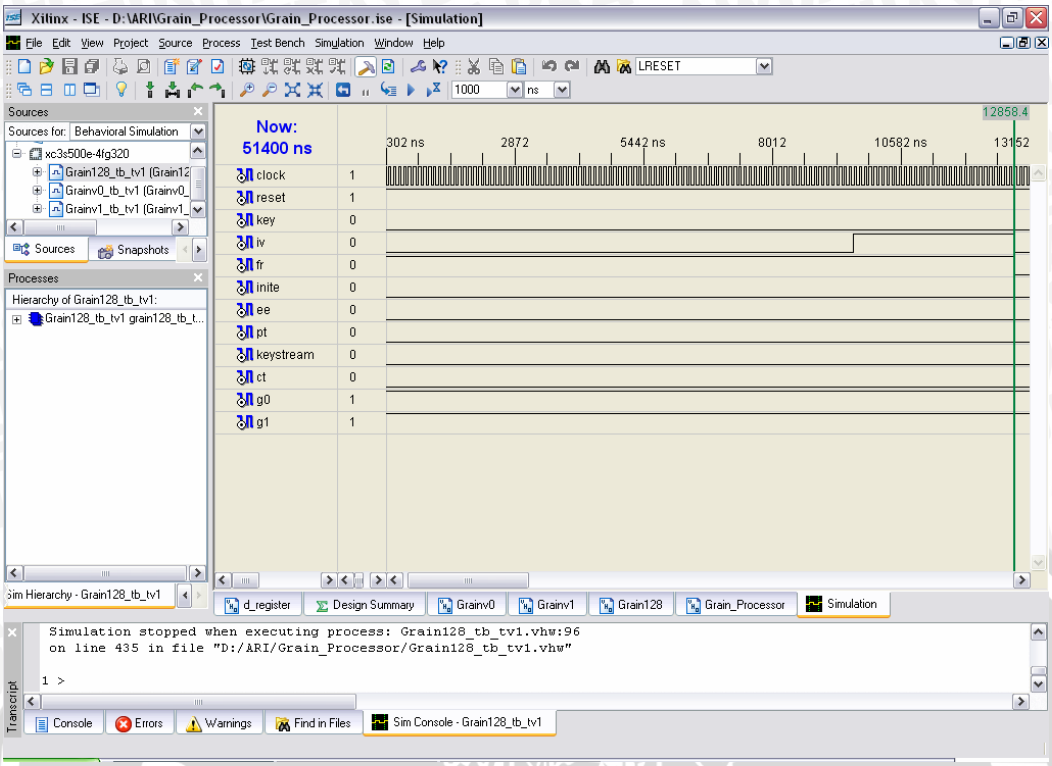


5.7 Hasil Simulasi Grain v1 Saat Pin `inite` dan `ee` Aktif Saat *Clock* ke 240

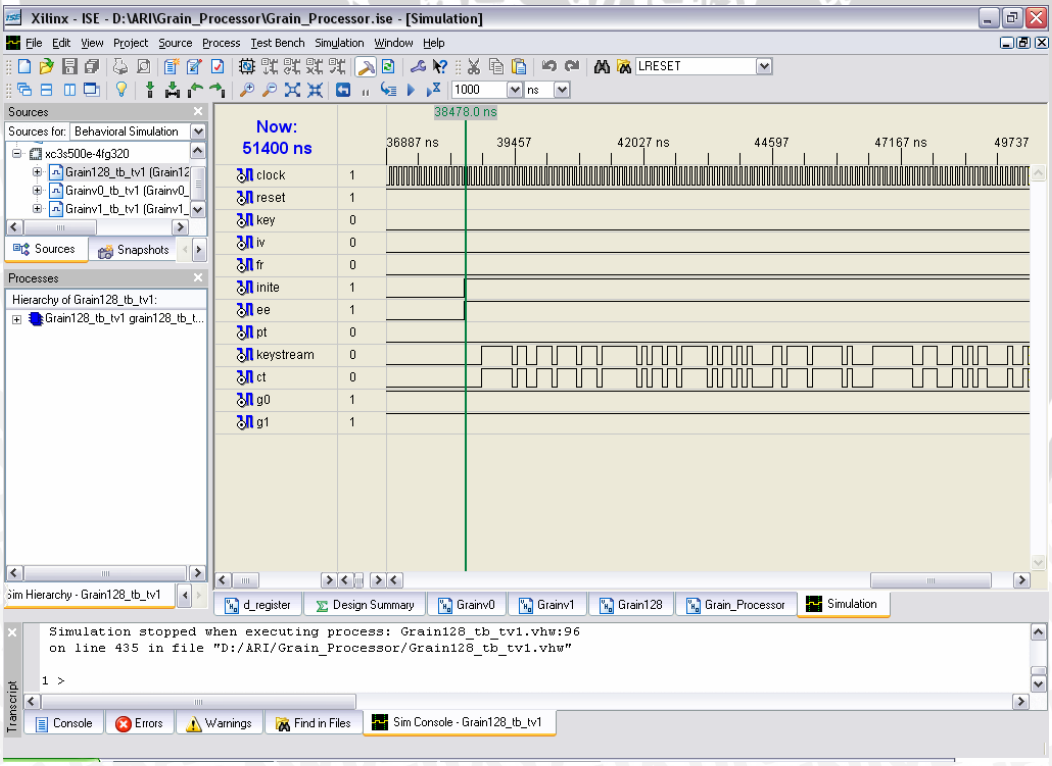
Sedikit berbeda dengan kedua versi Grain sebelumnya, Grain 128 ini memiliki panjang kunci dan panjang *initialization vector* yang berbeda, begitu pula proses inisialisasi kunci. Untuk memilih Grain 128 ini, pin `g0` diset berlogika 1 dan pin `g1` diset berlogika 1.

Untuk me-*load* kunci sebanyak 128 bit pada pin `key`, pin $f/\bar{r}$ diset berlogika 1 pada 128 *clock* pertama. Pada saat yang sama, 96 bit *initialization vector* di-*load* pada pin `iv` dengan 32 bit sisa diisi dengan logika 1. Berikutnya, proses inisialisasi kunci dilakukan sebanyak 256 *clock* tanpa menghasilkan *keystream*, dan akan berakhir saat pin `inite` diaktifkan pada *clock* ke 384.

Hasil *keystream* dapat dilihat melaui pin `keystream` dengan mengaktifkan pin `ee` pada saat *clock* ke 384, seperti dalam Gambar 5.9. Pin `pt` digunakan untuk me-*load* data yang akan dienkripsi pada saat *clock* ke 384. Dari pin `ct` dapat dilihat hasil dari proses enkripsi. Dengan *key* dan *iv* yang diberikan, hasil *keystream* tersebut telah sesuai dengan *test vector* yang telah diberikan oleh Grain 128. Hasil simulasi Grain 128 tersebut dapat dilihat pada Gambar 5.8 sampai 5.9 berikut ini.



Gambar 5.8 Hasil Simulasi Grain128 Saat Pin f / $\bar{r}$ Berlogika 0 Saat *Clock* ke 128



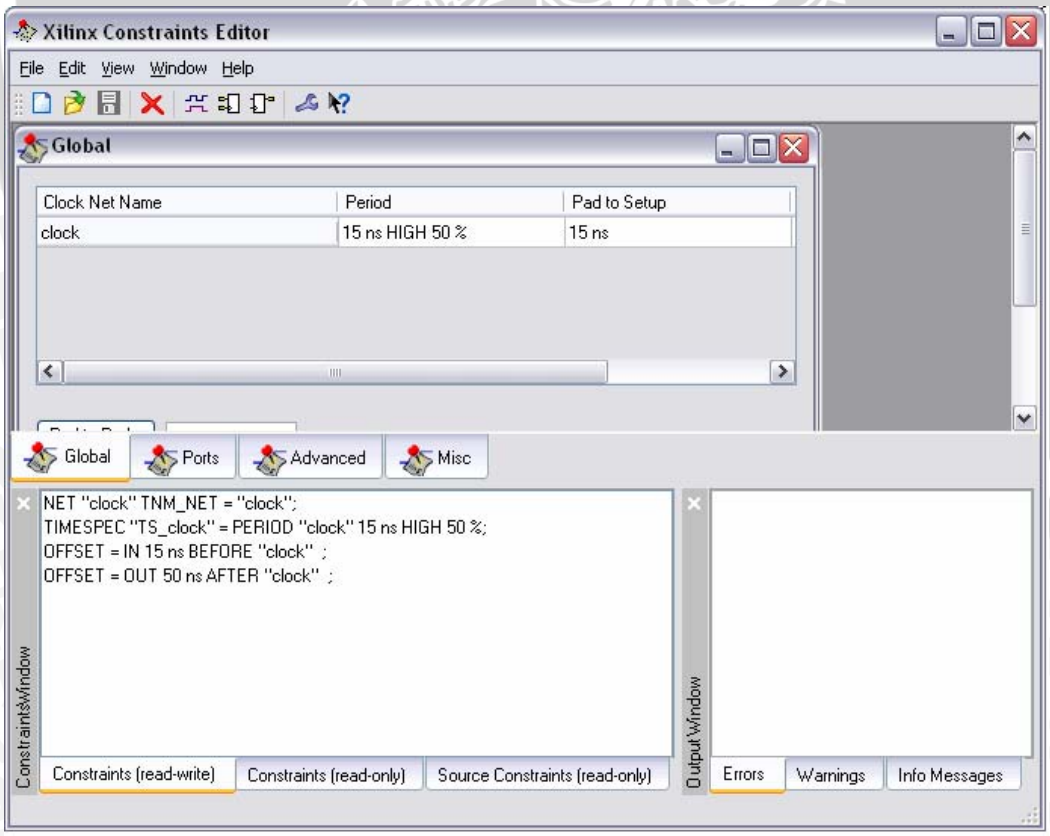
Gambar 5.9 Hasil Simulasi Grain128 Saat Pin `inite` dan `ee` Aktif Saat *Clock* ke 384

5.2.4 Tahap Implementasi

Tahap berikutnya adalah tahap implementasi yang terdiri dari tiga tahapan yaitu tahap *Translate*, *Map*, dan *Place and Route* yang akan dibahas lebih lanjut pada bagian ini.

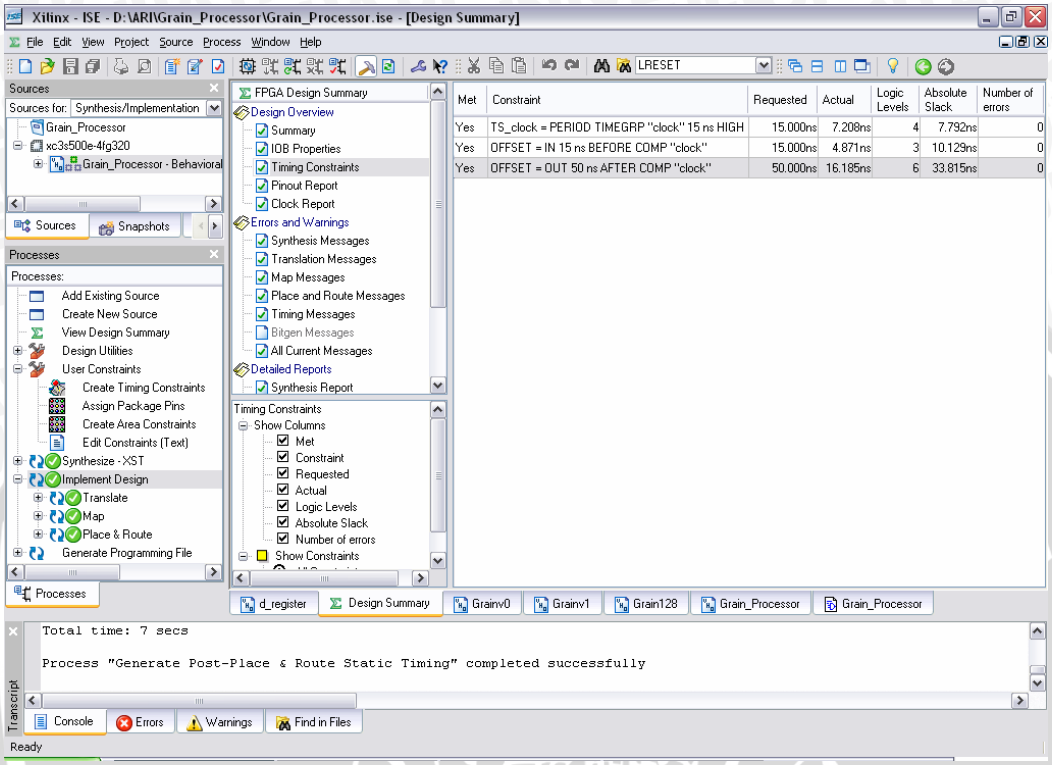
Untuk mengoptimalkan unjuk kerja pengimplementasian *stream cipher cryptography processor* algoritma Grain ini, perlu dilakukan spesifikasi *constraint* yang meliputi *constraint* waktu dan *constraint* pin. *Timing constraint* digunakan sebagai pedoman *place and route* pada desain. *Clock period* digunakan untuk menspesifikasikan *frequency* yang digunakan dalam pengoperasian FPGA.

Fasilitas yang digunakan untuk spesifikasi *constraint* ini adalah *Synthesis/Implementation* pada *Source Window* dan *User Constraint* pada *Process Window*. Selain *timing constraint* yang digunakan untuk *stream cipher cryptography processor* algoritma Grain ini, terdapat juga *pin assignment* yang digunakan untuk menentukan pin yang akan diimplementasikan pada FPGA yang digunakan. Pengesetan *constraint* tersebut dapat dilihat dalam Gambar 5.10.



Gambar 5.10 *Timing Constraint*

Jika semua *constraint* dapat dimplementasikan pada FPGA, maka akan muncul pesan bahwa tidak terjadi kesalahan. Hasil *constraint* tersebut dapat dilihat dalam Gambar 5.11.



Gambar 5.11 Hasil *Timing Constraint*

Tahap implementasi yang pertama dimulai dengan tahap *Translation* yang digunakan untuk mengimplementasikan *constraint* yang telah didefinisikan sebelumnya. Hasil dari proses ini berupa *file* `.ngd` dan `.bld`. *Translation Report* selengkapnya dapat dilihat pada Lampiran 3.

Tahap *Mapping* merupakan tahapan berikutnya yang merupakan proses untuk memetakan hasil *Translation* pada *device* yang digunakan. Hasil tahap *Mapping* ini berupa pemetaan gerbang-gerbang logika yang digunakan. Untuk pembuatan *stream cipher cryptography processor* algoritma Grain ini *Mapping Report* dapat dilihat pada Lampiran 4.

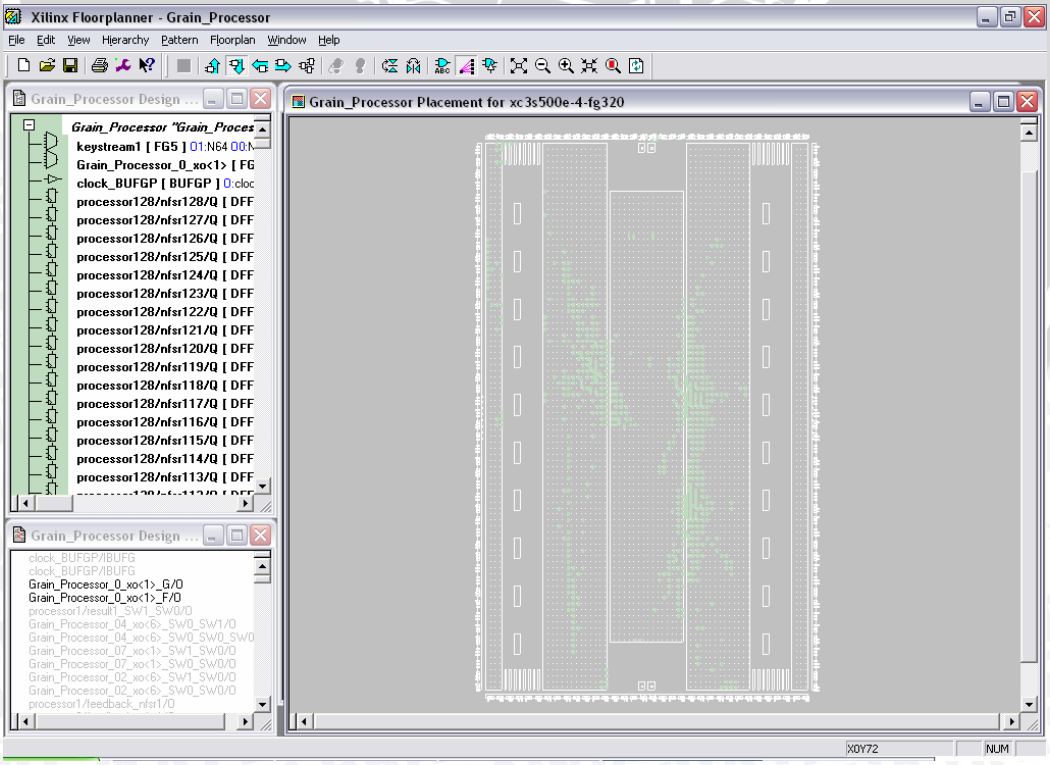
Tahapan terakhir dari tahap implementasi adalah *Place and Route* yang merupakan proses implementasi pada *chip*. *Place* adalah sebuah proses untuk memilih modul yang spesifik atau *logic stream* pada FPGA tempat gerbang logika akan ditempatkan. *Route* adalah proses *routing* fisik yang menghubungkan antar *logic stream*. Hasil *Place and Route* ini dapat dilihat pada Lampiran 5.

Tabel 5.1 adalah hasil dari pengimplementasian *stream cipher cryptography processor* algoritma Grain. *Floorplan* dari hasil proses *Place and Route stream cipher cryptography processor* algoritma Grain ini dapat dilihat dalam Gambar 5.12dan Gambar 5.13.

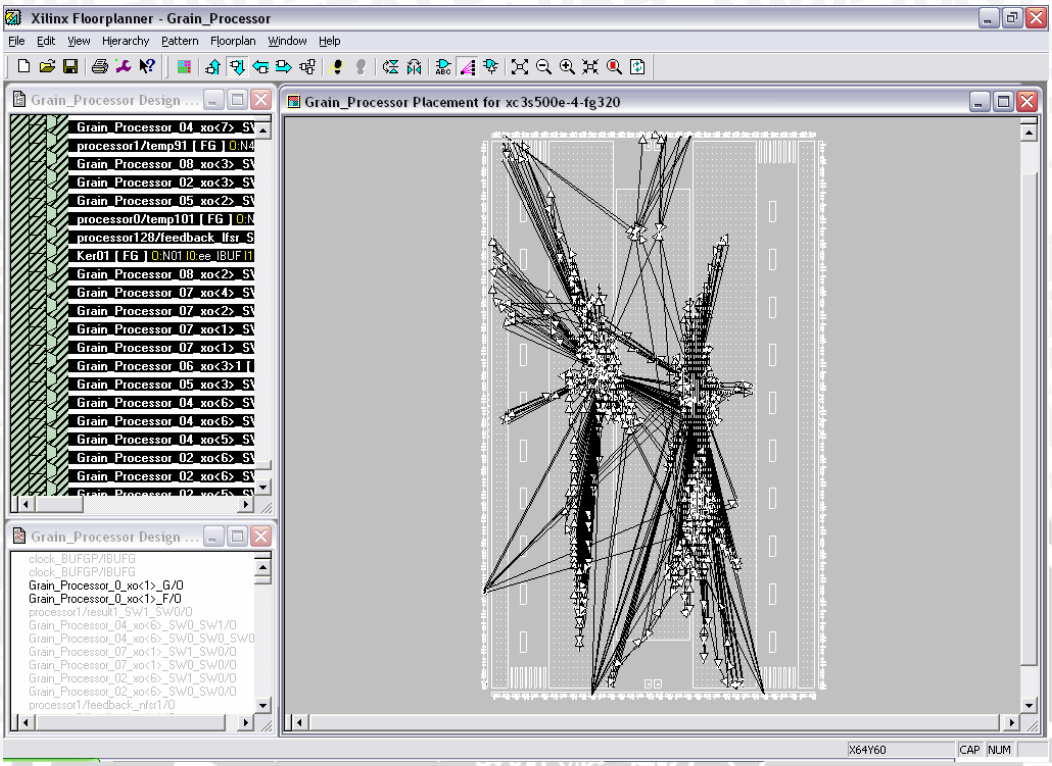
Tabel 5.1 *Design Summary*

GRAIN_PROCESSOR Project Status			
Project File:	Grain_Processor.ise	**Current State:**	Placed and Routed
Module Name:	Grain_Processor	**• Errors:**	No Errors
Target Device:	xc3s500e-4fg320	**• Warnings:**	No Warnings
Product Version:	ISE, 8.1i	**• Updated:**	Thu Nov 15 13:41:59 2007

Device Utilization Summary				
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	576	9,312	6%	
Number of 4 input LUTs	96	9,312	1%	
Logic Distribution				
Number of occupied Slices	625	4,656	13%	
Number of Slices containing only related logic	625	625	100%	
Number of Slices containing unrelated logic	0	625	0%	
Total Number of 4 input LUTs	96	9,312	1%	
Number of bonded IOBs	12	232	5%	
Number of GCLKs	1	24	4%	
Total equivalent gate count for design	5,193			
Additional JTAG gate count for IOBs	576			

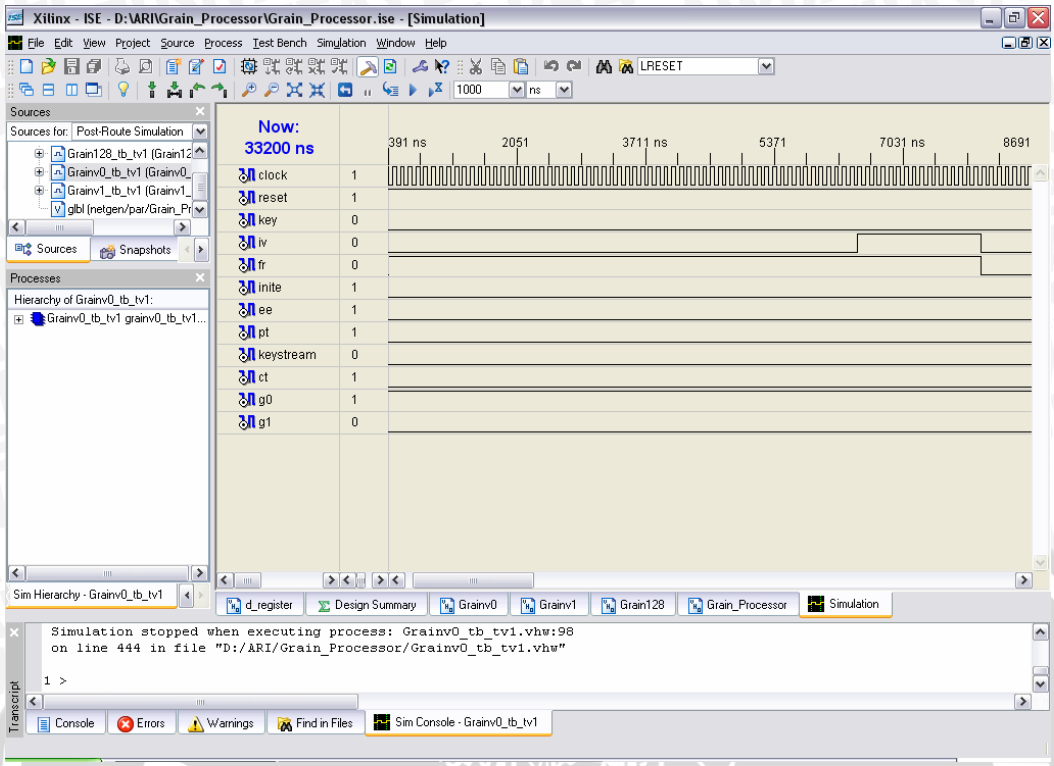


Gambar 5.12 *Floorplan* Grain *Processor*

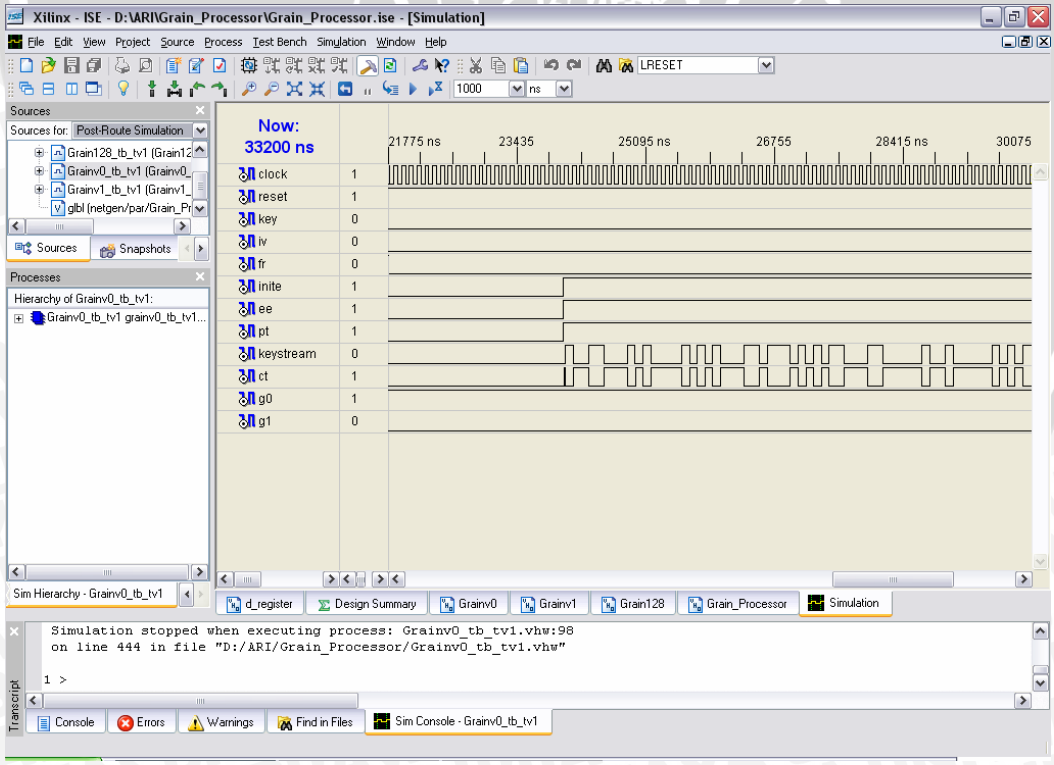


Gambar 5.13 *Routing Floorplan* Grain *Processor*

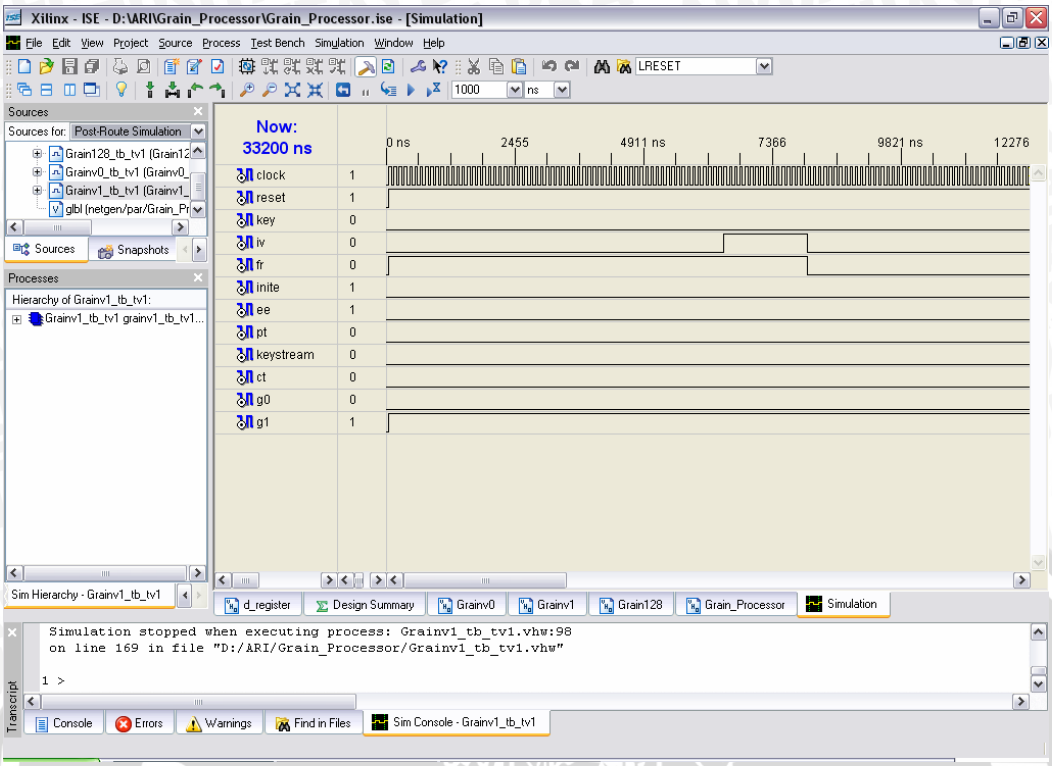
Setelah tahap implementasi berhasil, maka untuk menguji kebenaran implementasi desain dan *constraint* yang diberikan, perlu dilakukan proses simulasi sekali lagi dengan menggunakan fasilitas *Post-Route Simulation* pada *Source Window* dan *Simulate Post-Place & Route Model* pada *Process Window*. Gambar 5.13 sampai Gambar 5.18 menunjukkan hasil *Post-Route Simulation stream cipher cryptography processor* algoritma Grain.



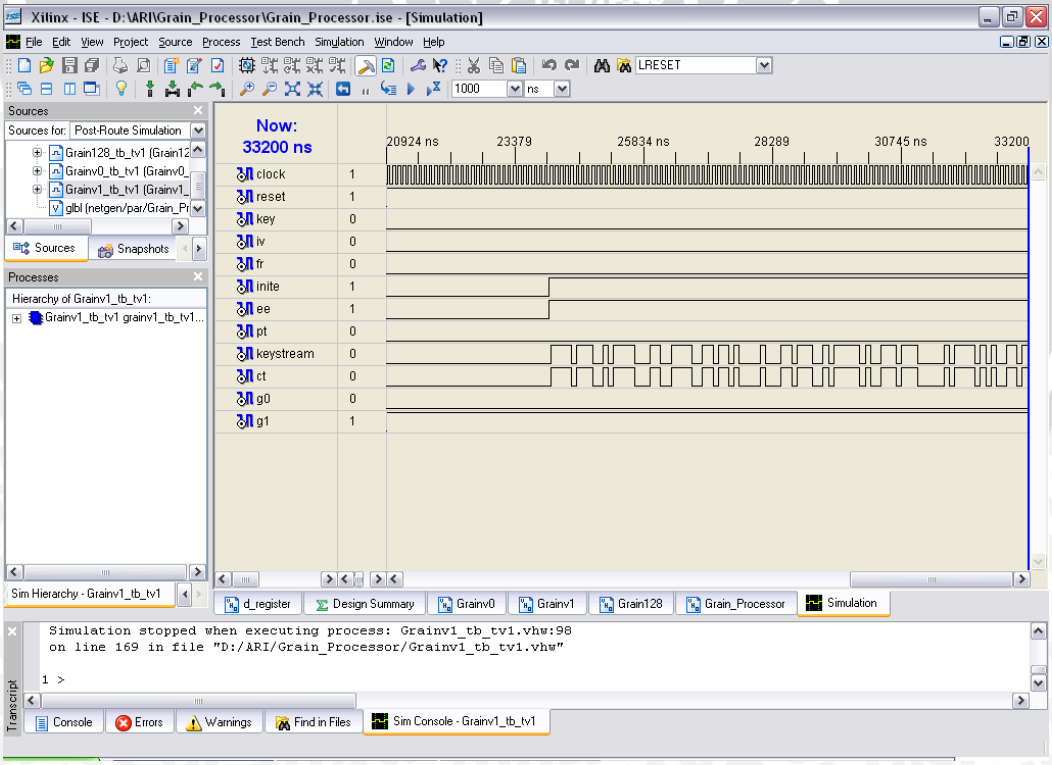
Gambar 5.13 Hasil *Post-Route Simulation* Grain v0 Saat Kunci dan *IV* Di-*load*



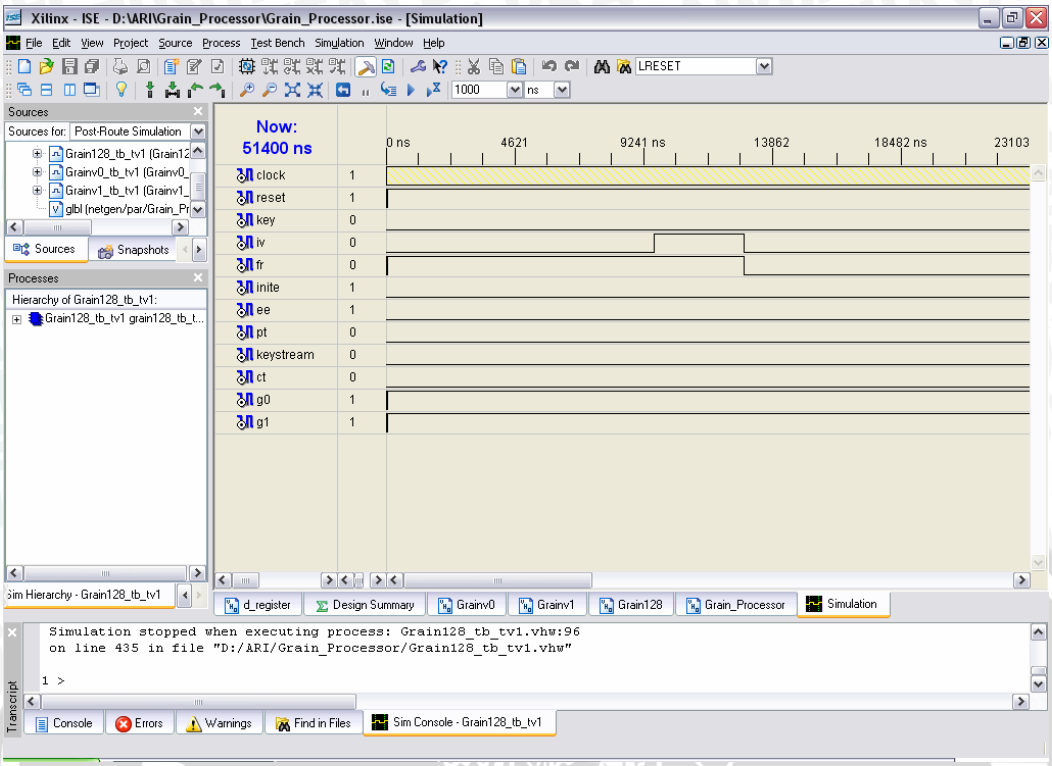
Gambar 5.14 Hasil *Post-Route Simulation* Grain v0 Saat *Keystream* Dihasilkan



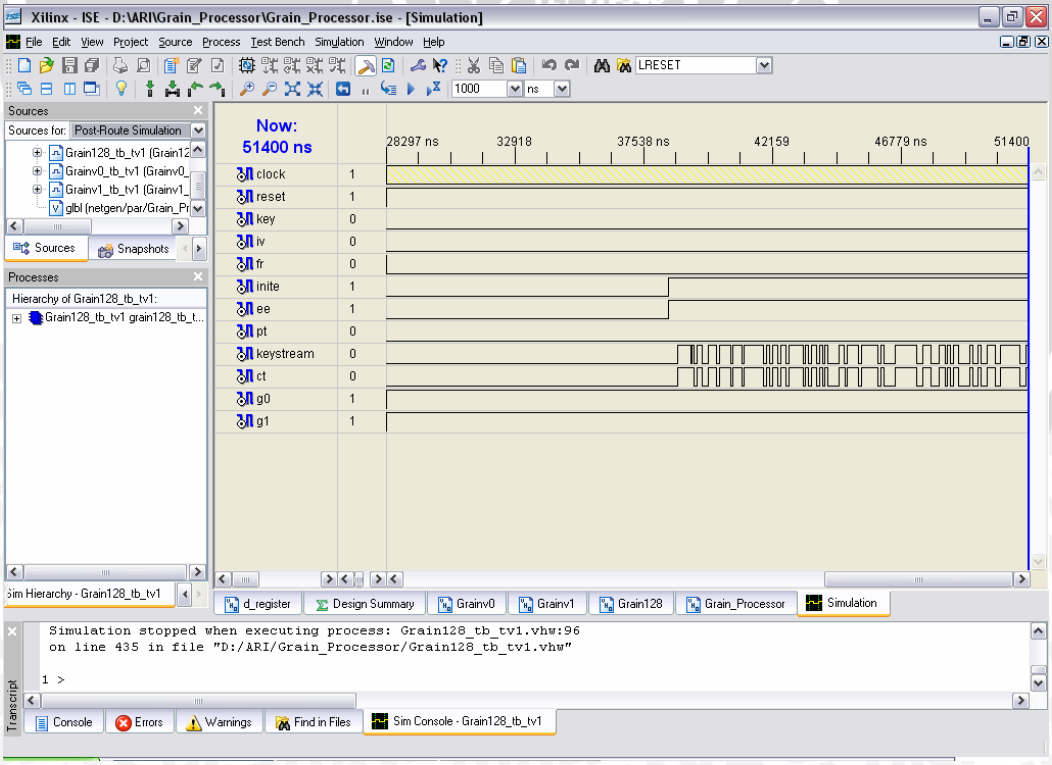
Gambar 5.15 Hasil *Post-Route Simulation* Grain v1 Saat Kunci dan *IV* Di-*load*



Gambar 5.16 Hasil *Post-Route Simulation* Grain v1 Saat *Keystream* Dihasilkan



Gambar 5.17 Hasil *Post-Route Simulation* Grain 128 Saat Kunci dan *IV* Di-*load*

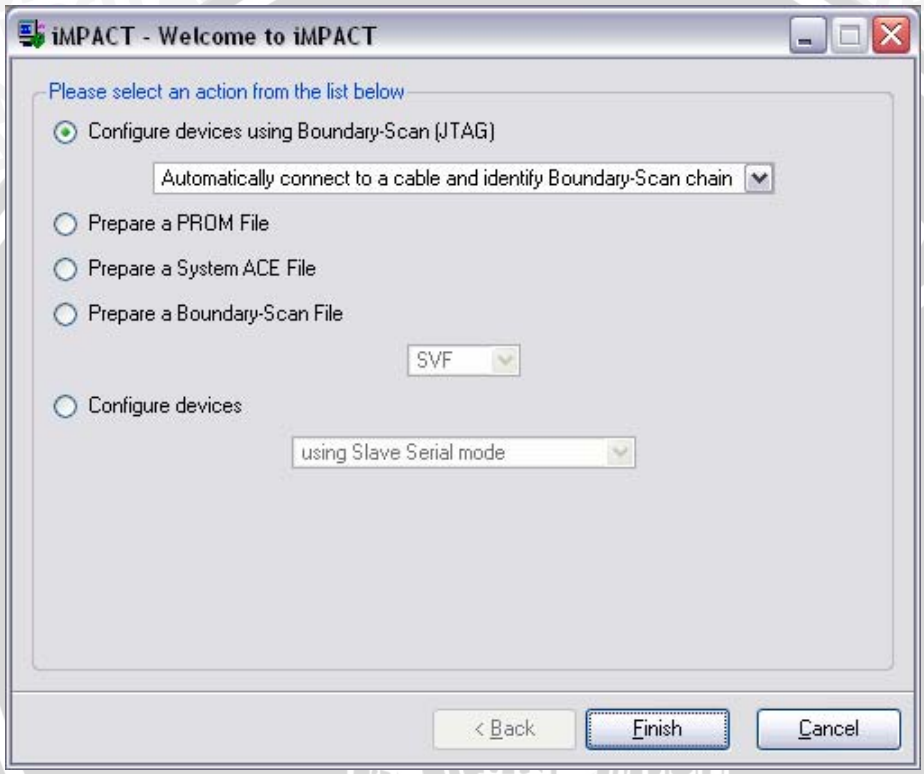


Gambar 5.18 Hasil *Post-Route Simulation* Grain 128 Saat *Keystream* Dihasilkan

5.2.5 Tahap *Programming*

Pada proses ini, hasil implementasi di-*download* pada FPGA dengan menggunakan fasilitas *Synthesis/Implementation* pada *Source Window* dan *Generate Programming File* serta *Configure Device (iMPACT)* pada *Process Window.*

Proses ini akan menampilkan kotak dialog iMPACT, dan pastikan menu yang dipilih adalah *Configure devices using Boundary-Scan (JTAG)* dan *Automatically connect to a cable and identify Boundary-Scan chain*, seperti dalam Gambar 5.19.



Gambar 5.19 *iMPACT*

Kotak dialog *Assign New Configuration File* akan muncul berikutnya, dan bila berhasil maka pesan akan muncul dan siap dijalankan pada FPGA.

BAB VI
PENGUJIAN

Setelah tahap implementasi, untuk mengetahui unjuk kerja dan validitas *stream cipher cryptography processor* algoritma Grain tersebut dilakukan tahap pengujian. Proses pengujian terdiri dari 2 macam, yaitu pengujian terhadap validitas *keystream* yang dihasilkan dan pengujian terhadap tingkat keacakan bit (*bit randomness*).

6.1 Pengujian Validitas *Keystream*

Metode pengujian ini dilakukan dengan simulasi menggunakan program *testbench* yaitu dengan menggunakan *test vector* Grain. *Test vector* yang digunakan adalah sebagai berikut.

6.1.1 *Test Vector* Grain v0

Key : 0000 0000 0000 0000 0000 H
IV : 0000 0000 0000 0000 H
Keystream : 4c50 a8dd 58c1 20ad 4c5b H

Key : 4444 4444 4444 4444 4444 H
IV : 5555 5555 5555 5555 H
Keystream : aa65 8ee6 45ef d34f 3428 H

Key : aaaa aaaa aaaa aaaa aaaa H
IV : bbbb bbbb bbbb bbbb H
Keystream : 99ac e85b 4be0 2846 b559 H

Key : 0123 4567 89ab cdef 1234 H
IV : 0123 4567 89ab cdef H
Keystream : 57f8 8839 0c05 684f a133 H

6.1.2 *Test Vector* Grain v1

Key : 0000 0000 0000 0000 0000 H
IV : 0000 0000 0000 0000 H
Keystream : 7b97 8cf3 6846 e5f4 ee0b H

Key : 0123 4567 89ab cdef 1234 H
IV : 0123 4567 89ab cdef H
Keystream : 42b5 67cc c653 1768 0225 H

Key : 4444 4444 4444 4444 4444 H
IV : 5555 5555 5555 5555 H
Keystream : db30 fdba 959c a9d8 69af H

Key : aaaa aaaa aaaa aaaa aaaa H
IV : bbbb bbbb bbbb bbbb H
Keystream : ff76 9dce df57 8046 2f96 H

6.1.3 *Test Vector* Grain 128

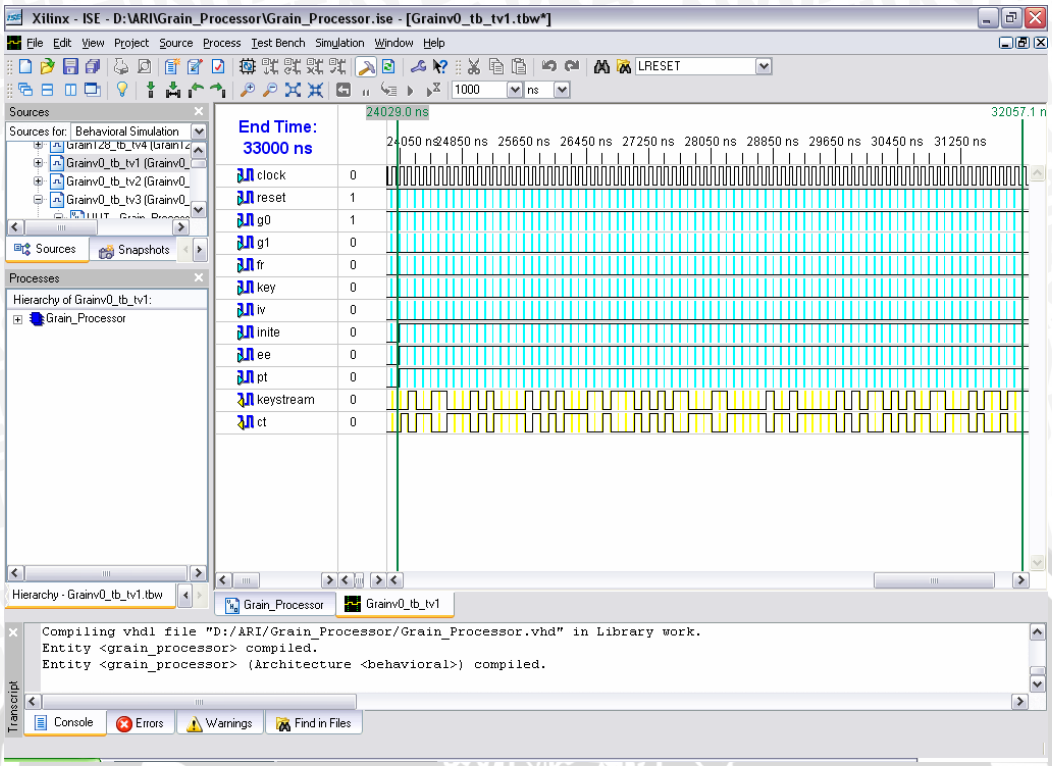
Key : 0000 0000 0000 0000 0000 0000 0000 0000 H
IV : 0000 0000 0000 0000 0000 0000 H
Keystream : 0fd9 deef eb6f ad43 7bf4 3fce 3584 9cfe H

Key : 0123 4567 89ab cdef 1234 5678 9abc def0 H
IV : 0123 4567 89ab cdef 1234 5678 H
Keystream : db03 2aff 3788 498b 57cb 894f ffb6 bb96 H

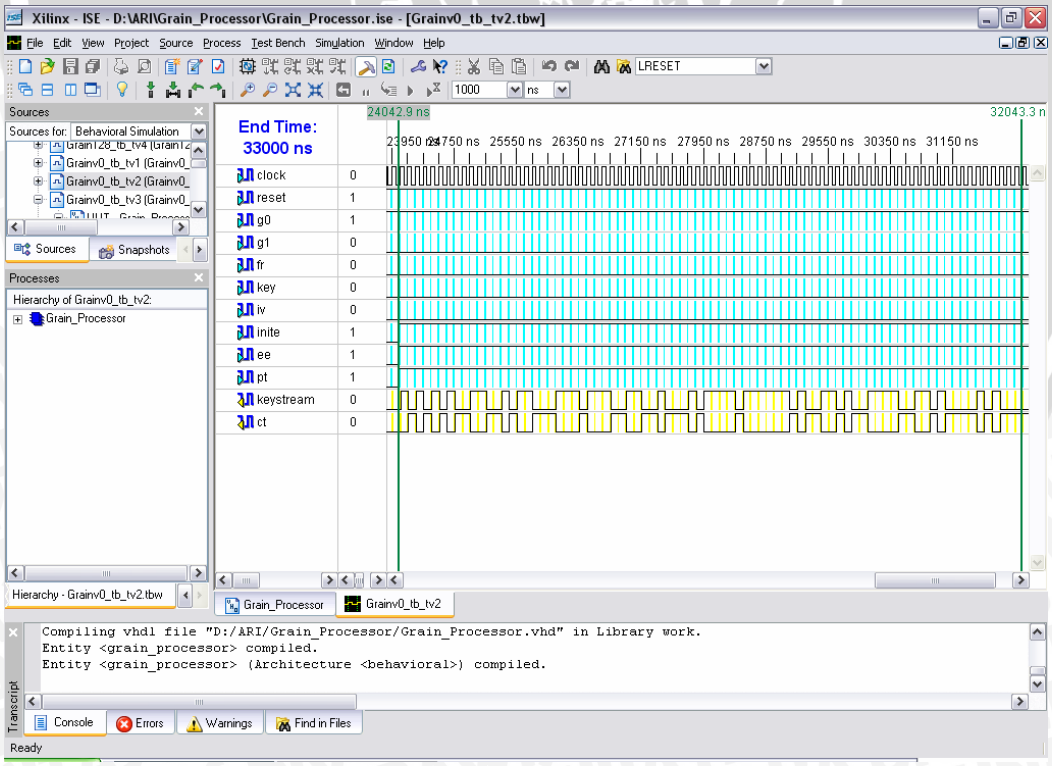
Key : 4444 4444 4444 4444 4444 4444 4444 4444 H
IV : 5555 5555 5555 5555 5555 5555 H
Keystream : 9386 2d3d 1b15 cbc6 e69c 1543 3ef7 eec4 H

Key : aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa H
IV : bbbb bbbb bbbb bbbb bbbb bbbb H
Keystream : e35e 58ec fb61 fba2 4b58 e8fc e220 33b5 H

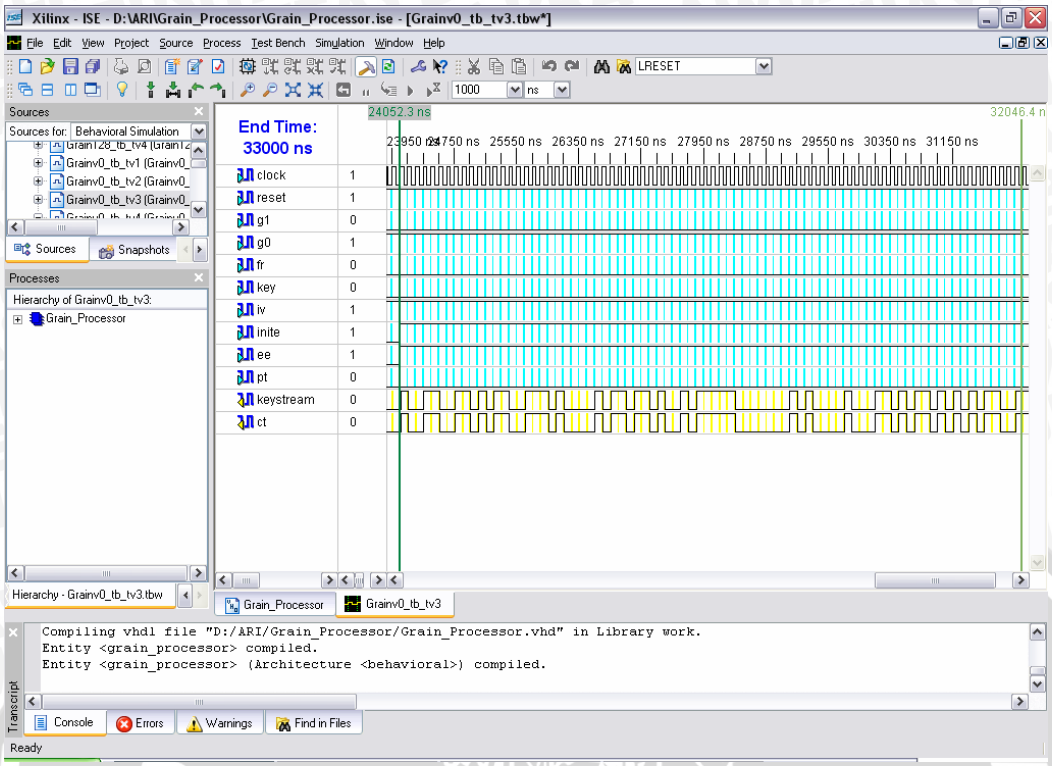
Hasil pengujian *keystream* utnuk masing-masing *test vector* Grain dapat dilihat dalam Gambar 6.1 sampai Gambar 6.12. Dari pengujian validitas *keystream* didapatkan bahwa *stream cipher cryptography processor* algoritma Grain ini telah berjalan sesuai dengan algoritma yang dibuat dalam tahap implementasi. Dan *keystream* yang dihasilkan telah sesuai dengan referensi *test vector* Grain yang diberikan oleh referensi [HJM-05] dan [HJM-06].



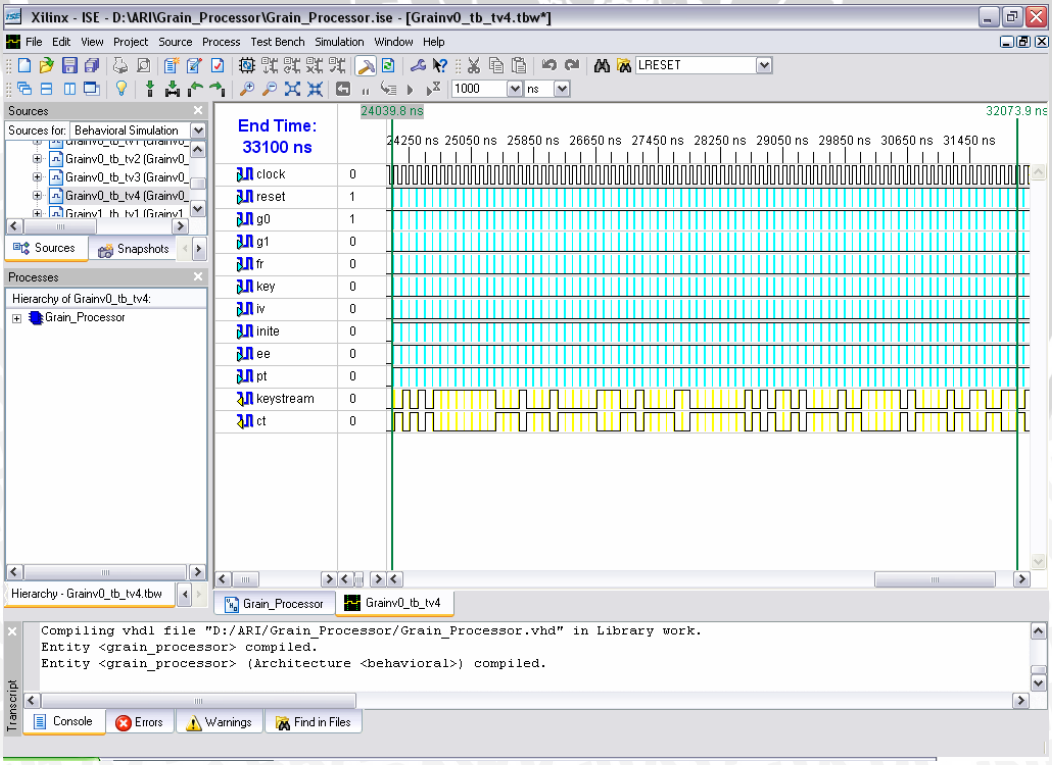
Gambar 6.1 Hasil Simulasi *Keystream* Grain v0 untuk *Test Vector* 1



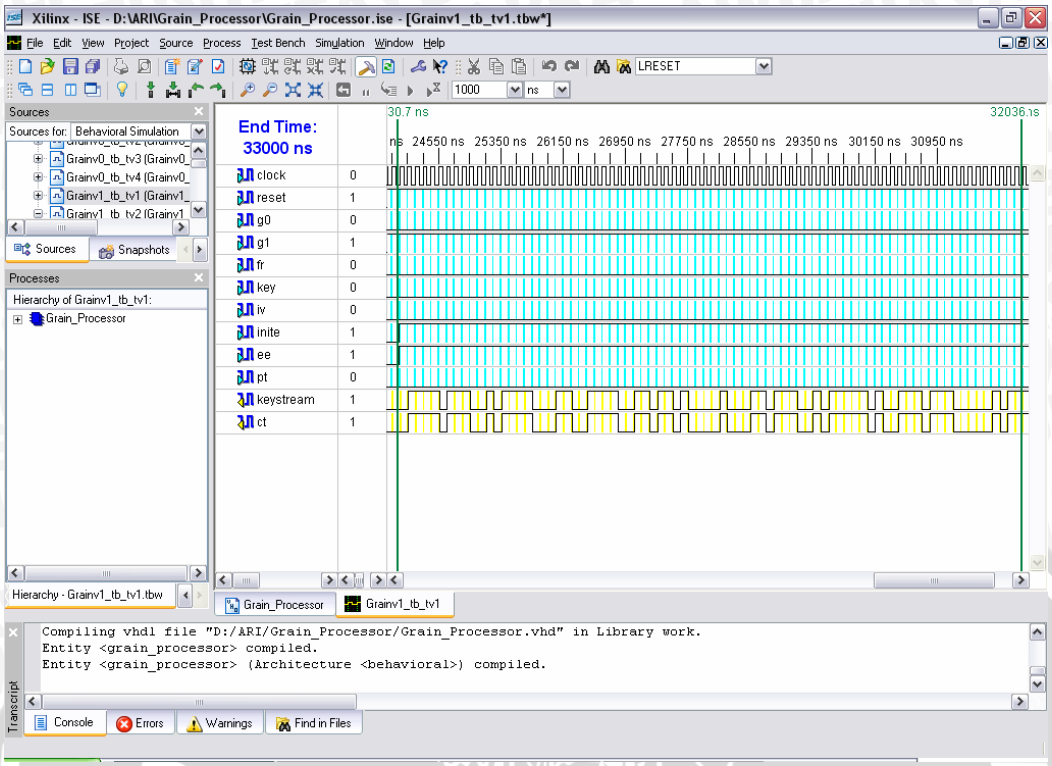
Gambar 6.2 Hasil Simulasi *Keystream* Grain v0 untuk *Test Vector* 2



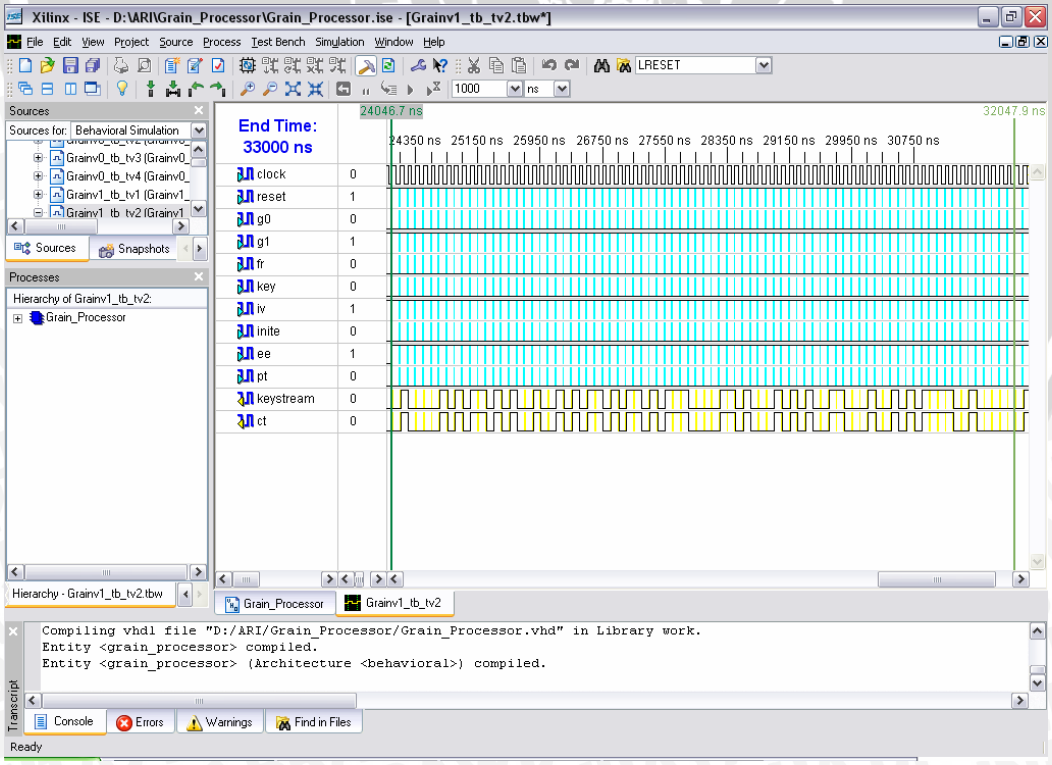
Gambar 6.3 Hasil Simulasi *Keystream* Grain v0 untuk *Test Vector* 3



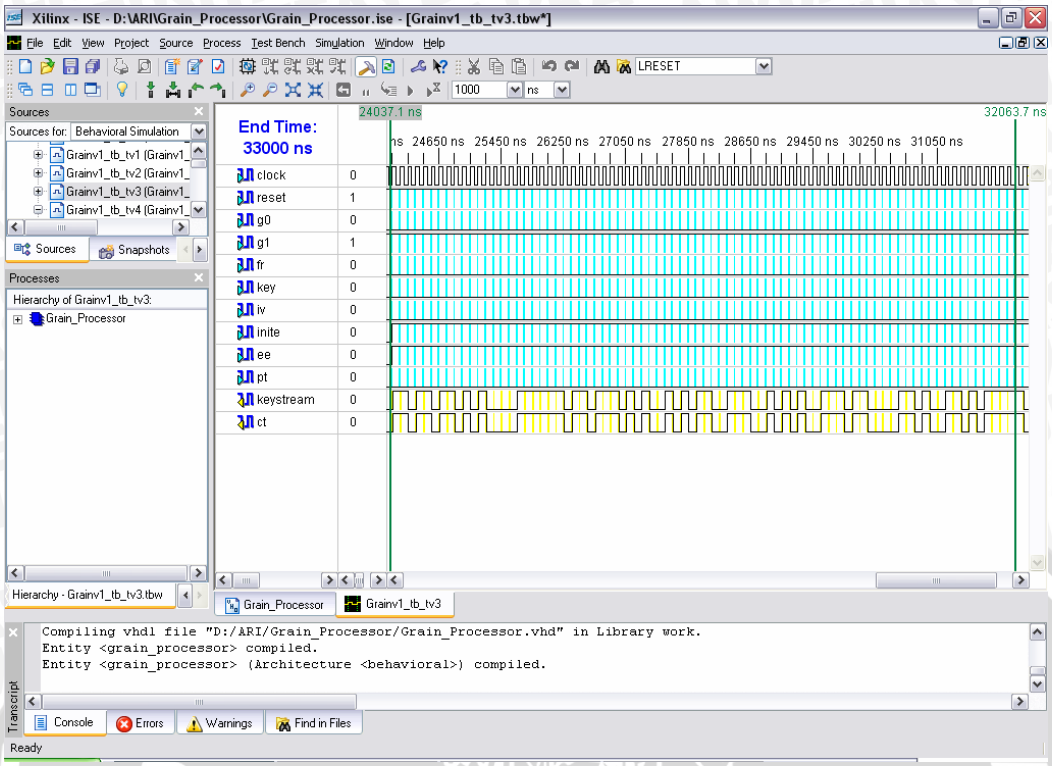
Gambar 6.4 Hasil Simulasi *Keystream* Grain v0 untuk *Test Vector* 4



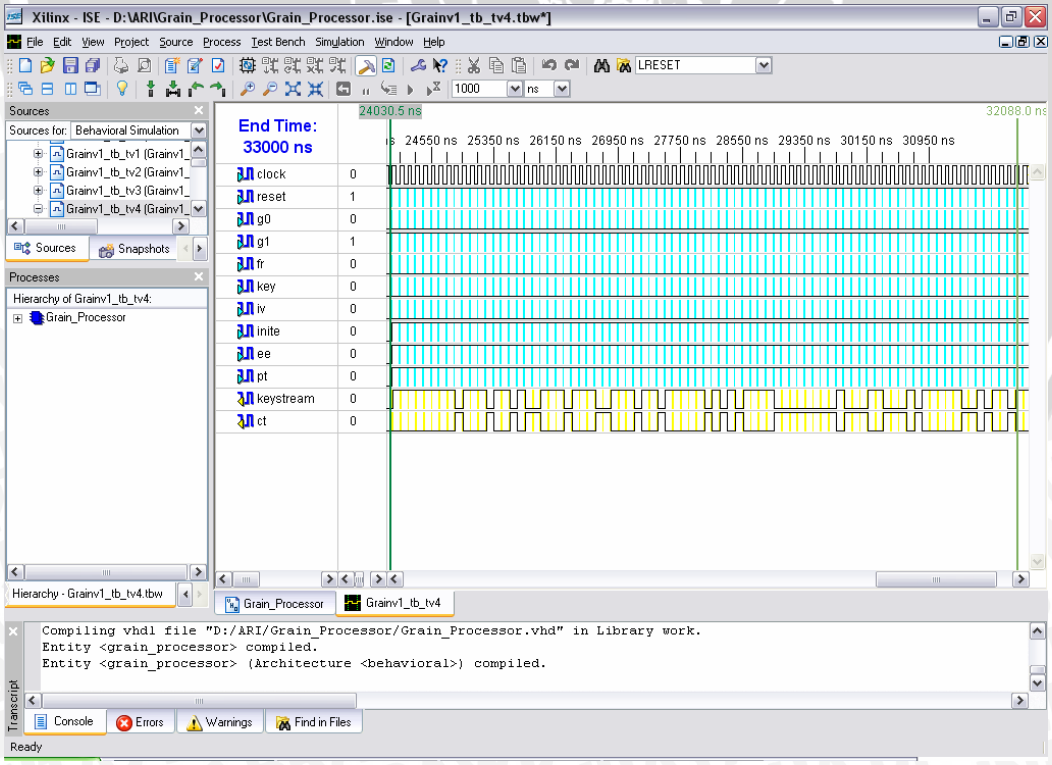
Gambar 6.5 Hasil Simulasi *Keystream* Grain v1 untuk *Test Vector* 1



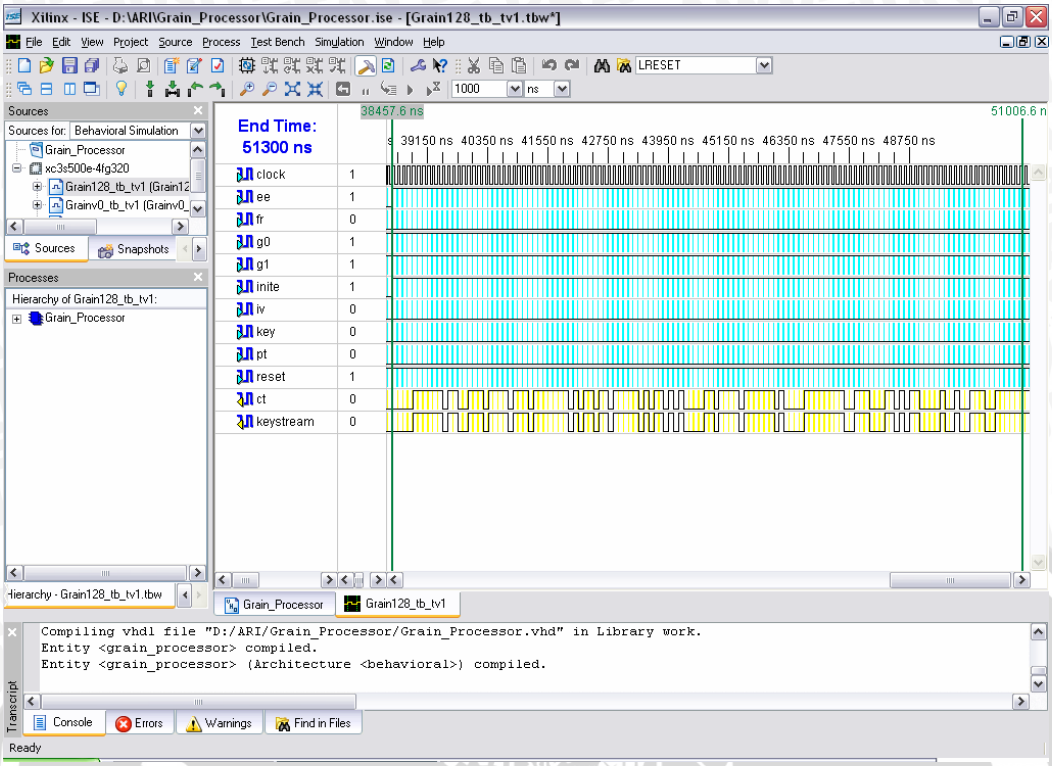
Gambar 6.6 Hasil Simulasi *Keystream* Grain v1 untuk *Test Vector* 2



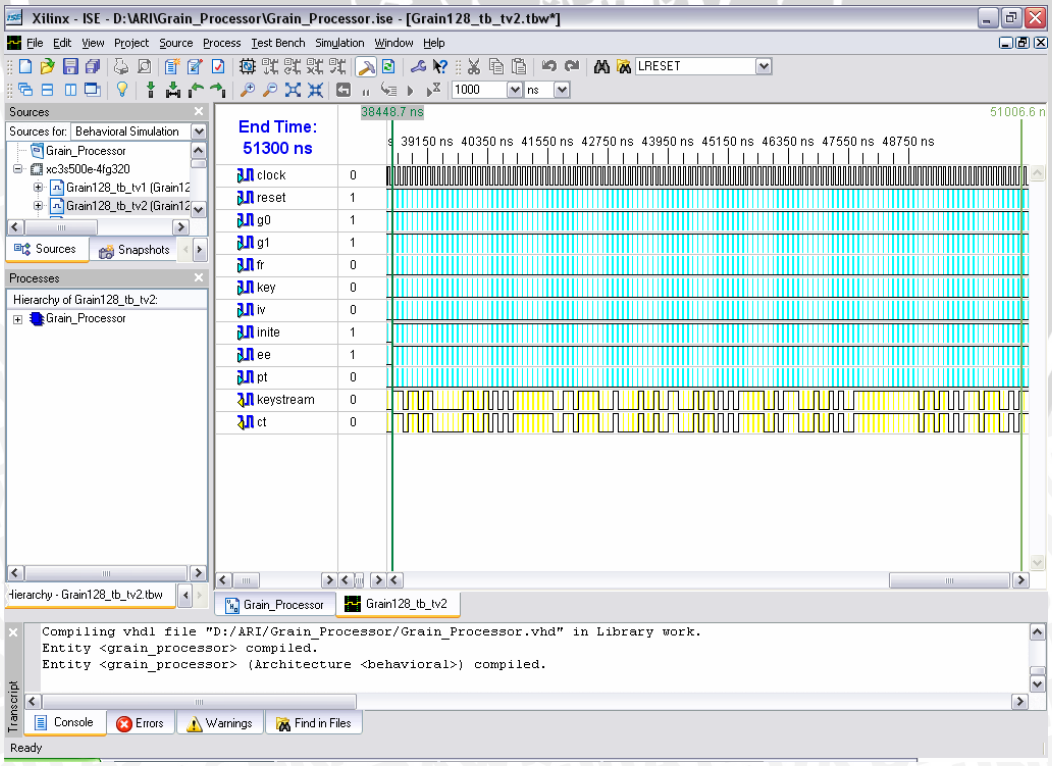
Gambar 6.7 Hasil Simulasi *Keystream* Grain v1 untuk *Test Vector* 3



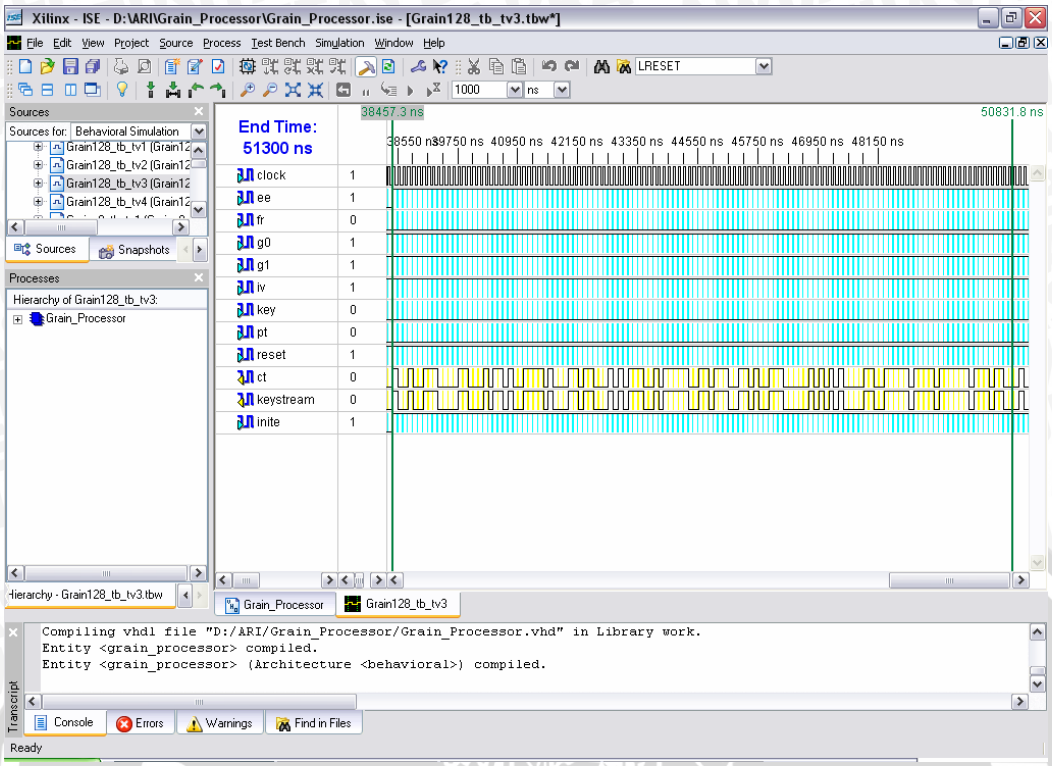
Gambar 6.8 Hasil Simulasi *Keystream* Grain v1 untuk *Test Vector* 4



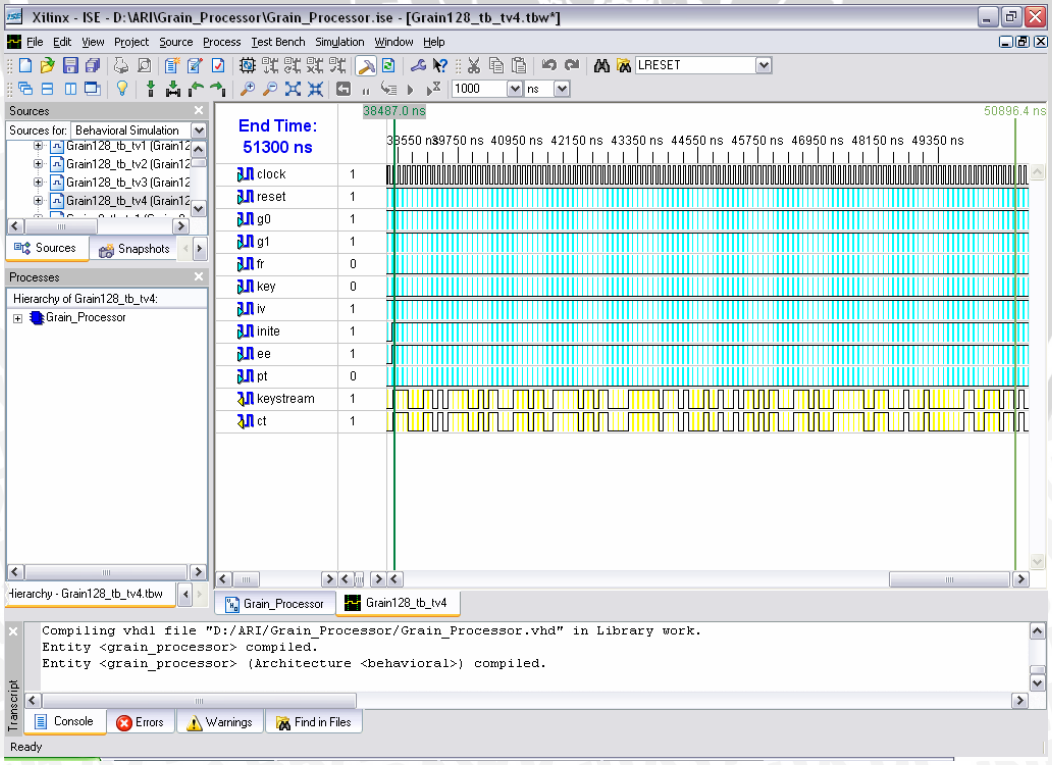
Gambar 6.9 Hasil Simulasi *Keystream* Grain 128 untuk *Test Vector* 1



Gambar 6.10 Hasil Simulasi *Keystream* Grain 128 untuk *Test Vector* 2



Gambar 6.11 Hasil Simulasi *Keystream* Grain 128 untuk *Test Vector* 3



Gambar 6.12 Hasil Simulasi *Keystream* Grain 128 untuk *Test Vector* 4

6.2 Pengujian Tingkat Keacakan Bit

Dalam pengujian tingkat keacakan bit akan dilakukan pengujian terhadap *keystream* yang dihasilkan. Pengujian ini didasarkan pada test standar yang dikeluarkan oleh *National Institute of Standard and Technology* [NIS-01], yang terdiri dari:

1. *Frequency Test*
2. *Frequency Test within a Block*
3. *The Runs Test*
4. *Test for the Longest Run of Ones in a Block*
5. *The Binary Matrix Rank Test*

Nilai *P-value* digunakan untuk menentukan tingkat keacakan suatu *sequence*. Jika nilai *P-value* lebih dari atau sama dengan 0,01, maka *sequence* tersebut dikatakan *random*. Hasil dari pengujian tersebut dapat dilihat pada Tabel 6.1 sampai Tabel 6.15. Untuk contoh tahap-tahap perhitungan setiap test dapat dilihat pada Lampiran 6.

6.2.1 Frequency Test

6.2.1.1 Grain v0

Tabel 6.1 Hasil *Frequency Test* Grain v0

No.	Key	IV	$\|S_{obs}\|$	*P-value*
1.	0000 0000 0000 0000 0000	0000 0000 0000 0000	11	0,2713
2.	8000 0000 0000 0000 0000	8000 0000 0000 0000	0	1
3.	C000 0000 0000 0000 0000	C000 0000 0000 0000	0	1
4.	E000 0000 0000 0000 0000	E000 0000 0000 0000	4	0,6892
5.	F000 0000 0000 0000 0000	F000 0000 0000 0000	1	0,3173
6.	F800 0000 0000 0000 0000	F800 0000 0000 0000	0	1
7.	FC00 0000 0000 0000 0000	FC00 0000 0000 0000	10	0,3173
8.	FE00 0000 0000 0000 0000	FE00 0000 0000 0000	4	0,6892
9.	FF00 0000 0000 0000 0000	FF00 0000 0000 0000	10	0,3173
10.	FF80 0000 0000 0000 0000	FF80 0000 0000 0000	10	0,3173

Dari Tabel 6.1 dapat dilihat bahwa hasil pengujian Grain v0, nilai *P-value* untuk *frequency test* memiliki yang lebih dari atau sama dengan 0,01. Sehingga dapat disimpulkan bahwa *sequence keystream* tersebut random.

6.2.1.2 Grain v1

Tabel 6.2 Hasil *Frequency Test* Grain v1

No.	Key	IV	$\|S_{obs}\|$	*P-value*
1.	0000 0000 0000 0000 0000	0000 0000 0000 0000	10	0,3173
2.	8000 0000 0000 0000 0000	8000 0000 0000 0000	4	0,6892
3.	C000 0000 0000 0000 0000	C000 0000 0000 0000	14	0,1615
4.	E000 0000 0000 0000 0000	E000 0000 0000 0000	6	0,5485
5.	F000 0000 0000 0000 0000	F000 0000 0000 0000	8	0,4237
6.	F800 0000 0000 0000 0000	F800 0000 0000 0000	18	0,0719
7.	FC00 0000 0000 0000 0000	FC00 0000 0000 0000	0	1
8.	FE00 0000 0000 0000 0000	FE00 0000 0000 0000	4	0,6892
9.	FF00 0000 0000 0000 0000	FF00 0000 0000 0000	4	0,6892
10.	FF80 0000 0000 0000 0000	FF80 0000 0000 0000	8	0,4237

Hasil pengujian Grain v1, nilai *P-value* untuk *frequency test* pada Tabel 6.2 memiliki nilai yang lebih dari atau sama dengan 0,01. Sehingga dapat disimpulkan bahwa *sequence keystream* tersebut random.

6.2.1.3 Grain 128

Tabel 6.3 Hasil *Frequency Test* Grain 128

No.	Key	IV	$\|S_{obs}\|$	*P-value*
1.	0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000	6	0,5485
2.	8000 0000 0000 0000 0000 0000 0000 0000	8000 0000 0000 0000 0000 0000 0000	16	0,11096
3.	C000 0000 0000 0000 0000 0000 0000 0000	C000 0000 0000 0000 0000 0000 0000	6	0,5485
4.	E000 0000 0000 0000 0000 0000 0000 0000	E000 0000 0000 0000 0000 0000 0000	8	0,4237
5.	F000 0000 0000 0000 0000 0000 0000 0000	F000 0000 0000 0000 0000 0000 0000	6	0,5485
6.	F800 0000 0000 0000 0000 0000 0000 0000	F800 0000 0000 0000 0000 0000 0000	8	0,4237
7.	FC00 0000 0000 0000 0000 0000 0000 0000	FC00 0000 0000 0000 0000 0000 0000	6	0,5485
8.	FE00 0000 0000 0000 0000 0000 0000 0000	FE00 0000 0000 0000 0000 0000 0000	10	0,3173
9.	FF00 0000 0000 0000 0000 0000 0000 0000	FF00 0000 0000 0000 0000 0000 0000	18	0,0119
10.	FF80 0000 0000 0000 0000 0000 0000 0000	FF80 0000 0000 0000 0000 0000 0000	4	0,6892

Nilai *P-value* untuk *frequency test* pengujian Grain 128 pada Tabel 6.3 memiliki nilai yang lebih dari atau sama dengan 0,01. Sehingga dapat disimpulkan bahwa *sequence keystream* tersebut random.

6.2.2 Frequency Test within a Block

6.2.2.1 Grain v0

Tabel 6.4 Hasil *Frequency Test within a Block* Grain v0

No.	Key	IV	X^2_{obs}	*P-value*
1.	0000 0000 0000 0000 0000	0000 0000 0000 0000	8,4	0,7016
2.	8000 0000 0000 0000 0000	8000 0000 0000 0000	15,6	0,1509
3.	C000 0000 0000 0000 0000	C000 0000 0000 0000	7,2	0,7970
4.	E000 0000 0000 0000 0000	E000 0000 0000 0000	2,4	0,9895
5.	F000 0000 0000 0000 0000	F000 0000 0000 0000	10,8	0,4874
6.	F800 0000 0000 0000 0000	F800 0000 0000 0000	9,2	0,6317
7.	FC00 0000 0000 0000 0000	FC00 0000 0000 0000	10	0,5595
8.	FE00 0000 0000 0000 0000	FE00 0000 0000 0000	10,8	0,4874
9.	FF00 0000 0000 0000 0000	FF00 0000 0000 0000	6,8	0,8253
10.	FF80 0000 0000 0000 0000	FF80 0000 0000 0000	13,2	0,2916

Dari hasil pengujian Grain v0 untuk *frequency test within a block* pada Tabel 6.4, nilai *P-value* memiliki nilai yang lebih dari atau sama dengan 0,01. Sehingga dapat disimpulkan bahwa *sequence keystream* tersebut random.

6.2.2.2 Grain v1

Tabel 6.5 Hasil *Frequency Test within a Block* Grain v1

No.	Key	IV	X^2_{obs}	*P-value*
1.	0000 0000 0000 0000 0000	0000 0000 0000 0000	12,4	0,3518
2.	8000 0000 0000 0000 0000	8000 0000 0000 0000	4	0,9596
3.	C000 0000 0000 0000 0000	C000 0000 0000 0000	9,6	0,5958
4.	E000 0000 0000 0000 0000	E000 0000 0000 0000	6	0,8753
5.	F000 0000 0000 0000 0000	F000 0000 0000 0000	12	0,3840
6.	F800 0000 0000 0000 0000	F800 0000 0000 0000	14,8	0,1909
7.	FC00 0000 0000 0000 0000	FC00 0000 0000 0000	12	0,3840
8.	FE00 0000 0000 0000 0000	FE00 0000 0000 0000	11,6	0,4175
9.	FF00 0000 0000 0000 0000	FF00 0000 0000 0000	9,6	0,5958
10.	FF80 0000 0000 0000 0000	FF80 0000 0000 0000	4	0,9596

Nilai *P-value* dari hasil pengujian Grain v1 untuk *frequency test within a block* pada Tabel 6.5 memiliki nilai yang lebih dari atau sama dengan 0,01. Sehingga dapat disimpulkan bahwa *sequence keystream* tersebut random.

6.2.2.3 Grain 128

Tabel 6.6 Hasil *Frequency Test within a Block* Grain 128

No.	Key	IV	X^2_{obs}	*P-value*
1.	0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000	15,2	0,1700
2.	8000 0000 0000 0000 0000 0000 0000 0000	8000 0000 0000 0000 0000 0000 0000	10,4	0,5233
3.	C000 0000 0000 0000 0000 0000 0000 0000	C000 0000 0000 0000 0000 0000 0000	11,6	0,4125
4.	E000 0000 0000 0000 0000 0000 0000 0000	E000 0000 0000 0000 0000 0000 0000	11,2	0,4520
5.	F000 0000 0000 0000 0000 0000 0000 0000	F000 0000 0000 0000 0000 0000 0000	8,8	0,6671
6.	F800 0000 0000 0000 0000 0000 0000 0000	F800 0000 0000 0000 0000 0000 0000	4	0,9596
7.	FC00 0000 0000 0000 0000 0000 0000 0000	FC00 0000 0000 0000 0000 0000 0000	8	0,7350
8.	FE00 0000 0000 0000 0000 0000 0000 0000	FE00 0000 0000 0000 0000 0000 0000	14,8	0,1909
9.	FF00 0000 0000 0000 0000 0000 0000 0000	FF00 0000 0000 0000 0000 0000 0000	11,6	0,4175
10.	FF80 0000 0000 0000 0000 0000 0000 0000	FF80 0000 0000 0000 0000 0000 0000	4	0,9596

Dari Tabel 6.6 hasil pengujian Grain 128, nilai *P-value* untuk *frequency test within a block* memiliki nilai yang lebih dari atau sama dengan 0,01. Sehingga dapat disimpulkan bahwa *sequence keystream* tersebut random.

6.2.3 Runs Test

6.2.3.1 Grain v0

Tabel 6.7 Hasil *Runs Test* Grain v0

No.	Key	IV	π	V_{obs}	*P-value*
1.	0000 0000 0000 0000 0000	0000 0000 0000 0000	0,43	50	0,8415
2.	8000 0000 0000 0000 0000	8000 0000 0000 0000	0,5	46	0,4237
3.	C000 0000 0000 0000 0000	C000 0000 0000 0000	0,5	56	0,2302
4.	E000 0000 0000 0000 0000	E000 0000 0000 0000	0,48	58	0,1055
5.	F000 0000 0000 0000 0000	F000 0000 0000 0000	0,53	52	0,6617
6.	F800 0000 0000 0000 0000	F800 0000 0000 0000	0,5	48	0,6892
7.	FC00 0000 0000 0000 0000	FC00 0000 0000 0000	0,52	52	0,6770
8.	FE00 0000 0000 0000 0000	FE00 0000 0000 0000	0,52	46	0,4323
9.	FF00 0000 0000 0000 0000	FF00 0000 0000 0000	0,45	56	0,1892
10.	FF80 0000 0000 0000 0000	FF80 0000 0000 0000	0,55	47	0,6135

Hasil pengujian Grain v0, nilai *P-value* untuk *runs test* pada Tabel 6.7 memiliki nilai yang lebih dari atau sama dengan 0,01. Sehingga dapat disimpulkan bahwa *sequence keystream* tersebut random.

6.2.3.2 Grain v1

Tabel 6.8 Hasil *Runs Test* Grain v1

No.	Key	IV	π	V_{obs}	*P-value*
1.	0000 0000 0000 0000 0000	0000 0000 0000 0000	0,55	45	0,3633
2.	8000 0000 0000 0000 0000	8000 0000 0000 0000	0,52	52	0,6770
3.	C000 0000 0000 0000 0000	C000 0000 0000 0000	0,57	52	0,5432
4.	E000 0000 0000 0000 0000	E000 0000 0000 0000	0,53	42	0,1165
5.	F000 0000 0000 0000 0000	F000 0000 0000 0000	0,54	51	0,7904
6.	F800 0000 0000 0000 0000	F800 0000 0000 0000	0,59	48	0,9374
7.	FC00 0000 0000 0000 0000	FC00 0000 0000 0000	0,5	40	0,0455
8.	FE00 0000 0000 0000 0000	FE00 0000 0000 0000	0,48	53	0,5372
9.	FF00 0000 0000 0000 0000	FF00 0000 0000 0000	0,52	48	0,7005
10.	FF80 0000 0000 0000 0000	FF80 0000 0000 0000	0,46	52	0,6405

Nilai *P-value* untuk *runs test* dari hasil pengujian Grain v1 pada Tabel 6.8, memiliki nilai yang lebih dari atau sama dengan 0,01. Sehingga dapat disimpulkan bahwa *sequence keystream* tersebut random.

6.2.3.3 Grain 128

Tabel 6.9 Hasil *Runs Test* Grain 128

No.	Key	IV	π	V_{obs}	*P-value*
1.	0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000	0,66	42	0,5210
2.	8000 0000 0000 0000 0000 0000 0000 0000	8000 0000 0000 0000 0000 0000 0000	0,42	52	0,5008
3.	C000 0000 0000 0000 0000 0000 0000 0000	C000 0000 0000 0000 0000 0000 0000	0,53	50	0,5485
4.	E000 0000 0000 0000 0000 0000 0000 0000	E000 0000 0000 0000 0000 0000 0000	0,46	50	0,9486
5.	F000 0000 0000 0000 0000 0000 0000 0000	F000 0000 0000 0000 0000 0000 0000	0,53	48	0,7149
6.	F800 0000 0000 0000 0000 0000 0000 0000	F800 0000 0000 0000 0000 0000 0000	0,54	56	0,2033
7.	FC00 0000 0000 0000 0000 0000 0000 0000	FC00 0000 0000 0000 0000 0000 0000	0,47	45	0,333
8.	FE00 0000 0000 0000 0000 0000 0000 0000	FE00 0000 0000 0000 0000 0000 0000	0,55	52	0,6135
9.	FF00 0000 0000 0000 0000 0000 0000 0000	FF00 0000 0000 0000 0000 0000 0000	0,41	56	0,1152
10.	FF80 0000 0000 0000 0000 0000 0000 0000	FF80 0000 0000 0000 0000 0000 0000	0,58	56	0,1351

Hasil pengujian Grain 128, nilai *P-value* untuk *runs test* pada Tabel 6.9 memiliki nilai yang lebih dari atau sama dengan 0,01. Sehingga dapat disimpulkan bahwa *sequence keystream* tersebut random.

6.2.4 *Test for the Longest Run of Ones in a Block*

6.2.4.1 Grain v0

Tabel 6.10 Hasil *Test for the Longest Run of Ones in a Block* Grain v0

No	Key	IV	v_0	v_1	v_2	v_3	X^2_{obs}	*P-value*
1.	0000 0000 0000 0000 0000	0000 0000 0000 0000	5	8	2	1	3,5854	0,5091
2.	8000 0000 0000 0000 0000	8000 0000 0000 0000	3	3	6	4	3,2453	0,5663
3.	C000 0000 0000 0000 0000	C000 0000 0000 0000	5	5	4	2	1,2011	0,8924
4.	E000 0000 0000 0000 0000	E000 0000 0000 0000	2	9	5	0	5,7294	0,2172
5.	F000 0000 0000 0000 0000	F000 0000 0000 0000	2	9	2	3	3,0352	0,6023
6.	F800 0000 0000 0000 0000	F800 0000 0000 0000	4	5	6	1	3,0054	0,6074
7.	FC00 0000 0000 0000 0000	FC00 0000 0000 0000	4	6	4	2	0,4547	0,9703
8.	FE00 0000 0000 0000 0000	FE00 0000 0000 0000	3	4	7	2	3,9617	0,4482
9.	FF00 0000 0000 0000 0000	FF00 0000 0000 0000	3	10	2	1	5,0573	0,2928
10.	FF80 0000 0000 0000 0000	FF80 0000 0000 0000	3	8	1	4	3,1164	0,5884

Untuk *test for the longest run of ones in a block*, dari Tabel 6.10 dapat dilihat bahwa hasil pengujian Grain v0, nilai *P-value* memiliki nilai yang lebih dari atau sama dengan 0,01. Sehingga dapat disimpulkan bahwa *sequence keystream* tersebut random.

6.2.4.2 Grain v1

Tabel 6.11 Hasil *Test for the Longest Run of Ones in a Block* Grain v1

No.	Key	IV	v_0	v_1	v_2	v_3	X^2_{obs}	*P-value*
1.	0000 0000 0000 0000 0000	0000 0000 0000 0000	2	7	3	4	1,2777	0,8826
2.	8000 0000 0000 0000 0000	8000 0000 0000 0000	3	7	3	3	0,3992	0,9747
3.	C000 0000 0000 0000 0000	C000 0000 0000 0000	1	9	3	3	3,5181	0,5203
4.	E000 0000 0000 0000 0000	E000 0000 0000 0000	2	2	9	3	10,8078	0,0106
5.	F000 0000 0000 0000 0000	F000 0000 0000 0000	2	8	4	2	1,7288	0,8190
6.	F800 0000 0000 0000 0000	F800 0000 0000 0000	1	9	2	4	4,4957	0,3678
7.	FC00 0000 0000 0000 0000	FC00 0000 0000 0000	2	5	6	3	2,1804	0,7476
8.	FE00 0000 0000 0000 0000	FE00 0000 0000 0000	5	5	3	3	0,9697	0,9202
9.	FF00 0000 0000 0000 0000	FF00 0000 0000 0000	3	5	5	3	0,6526	0,9531
10.	FF80 0000 0000 0000 0000	FF80 0000 0000 0000	4	8	3	1	3,3224	0,7241

Dari Tabel 6.11 hasil pengujian Grain v1, nilai *P-value* untuk *test for the longest run of ones in a block* memiliki nilai yang lebih dari atau sama dengan 0,01. Sehingga dapat disimpulkan bahwa *sequence keystream* tersebut random.

6.2.4.3 Grain 128

Tabel 6.12 Hasil *Test for the Longest Run of Ones in a Block* Grain 128

No	Key	IV	v_0	v_1	v_2	v_3	X^2_{obs}	*P-value*
1.	0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000	1	4	3	8	10,7880	0,0108
2.	8000 0000 0000 0000 0000 0000 0000 0000	8000 0000 0000 0000 0000 0000 0000	4	8	3	1	3,3224	0,5532
3.	C000 0000 0000 0000 0000 0000 0000 0000	C000 0000 0000 0000 0000 0000 0000	3	6	5	2	0,8582	0,9325
4.	E000 0000 0000 0000 0000 0000 0000 0000	E000 0000 0000 0000 0000 0000 0000	7	3	2	4	6,2072	0,1728
5.	F000 0000 0000 0000 0000 0000 0000 0000	F000 0000 0000 0000 0000 0000 0000	5	6	2	3	1,4863	0,8543
6.	F800 0000 0000 0000 0000 0000 0000 0000	F800 0000 0000 0000 0000 0000 0000	2	7	4	3	0,8424	0,9341
7.	FC00 0000 0000 0000 0000 0000 0000 0000	FC00 0000 0000 0000 0000 0000 0000	3	6	5	2	0,8582	0,9325
8.	FE00 0000 0000 0000 0000 0000 0000 0000	FE00 0000 0000 0000 0000 0000 0000	6	3	4	3	3,3451	0,5494
9.	FF00 0000 0000 0000 0000 0000 0000 0000	FF00 0000 0000 0000 0000 0000 0000	8	3	3	2	7,9275	0,0685
10.	FF80 0000 0000 0000 0000 0000 0000 0000	FF80 0000 0000 0000 0000 0000 0000	5	8	2	1	3,5854	0,5091

Nilai *P-value* untuk *test for the longest run of ones in a block* dari hasil pengujian Grain 128 pada Tabel 6.12 memiliki nilai yang lebih dari atau sama dengan 0,01. Sehingga dapat disimpulkan bahwa *sequence keystream* tersebut random.

6.2.5 Binary Matrix Rank Test

6.2.5.1 Grain v0

Tabel 6.13 Hasil *Binary Matrix Rank Test* Grain v0

No	Key	IV	F_M	F_{M-1}	N	X^2_{obs}	*P-value*
1.	0000 0000 0000 0000 0000	0000 0000 0000 0000	9	24	5	0,7753	0,8850
2.	8000 0000 0000 0000 0000	8000 0000 0000 0000	12	21	5	0,1380	0,9838
3.	C000 0000 0000 0000 0000	C000 0000 0000 0000	7	26	5	2,1882	0,5917
4.	E000 0000 0000 0000 0000	E000 0000 0000 0000	14	18	6	1,7125	0,6943
5.	F000 0000 0000 0000 0000	F000 0000 0000 0000	11	22	5	0,0013	0,9999
6.	F800 0000 0000 0000 0000	F800 0000 0000 0000	12	22	4	0,3244	0,9588
7.	FC00 0000 0000 0000 0000	FC00 0000 0000 0000	14	17	7	2,6785	0,4899
8.	FE00 0000 0000 0000 0000	FE00 0000 0000 0000	10	26	2	2,6990	0,4859
9.	FF00 0000 0000 0000 0000	FF00 0000 0000 0000	13	20	5	0,5481	0,9243
10.	FF80 0000 0000 0000 0000	FF80 0000 0000 0000	9	24	5	0,5481	0,9243

Nilai *P-value* untuk *binary matrix rank test* dari hasil pengujian Grain v0 pada Tabel 6.13 memiliki nilai yang lebih dari atau sama dengan 0,01. Sehingga dapat disimpulkan bahwa *sequence keystream* tersebut random.

6.2.5.2 Grain v1

Tabel 6.14 Hasil *Binary Matrix Rank Test* Grain v1

No	Key	IV	F_M	F_{M-1}	N	X^2_{obs}	*P-value*
1.	0000 0000 0000 0000 0000	0000 0000 0000 0000	17	21	0	8,4262	0,0136
2.	8000 0000 0000 0000 0000	8000 0000 0000 0000	12	24	2	2,1522	0,5994
3.	C000 0000 0000 0000 0000	C000 0000 0000 0000	15	21	2	3,3824	0,3593
4.	E000 0000 0000 0000 0000	E000 0000 0000 0000	13	20	5	0,5481	0,9243
5.	F000 0000 0000 0000 0000	F000 0000 0000 0000	14	18	6	1,7125	0,6943
6.	F800 0000 0000 0000 0000	F800 0000 0000 0000	13	22	3	1,2236	0,7979
7.	FC00 0000 0000 0000 0000	FC00 0000 0000 0000	16	18	4	3,2402	0,3839
8.	FE00 0000 0000 0000 0000	FE00 0000 0000 0000	16	19	3	3,5472	0,3320
9.	FF00 0000 0000 0000 0000	FF00 0000 0000 0000	17	18	3	4,8684	0,1631
10.	FF80 0000 0000 0000 0000	FF80 0000 0000 0000	15	21	2	3,3824	0,3593

Untuk *binary matrix rank test*, dari Tabel 6.14 dapat dilihat bahwa hasil pengujian Grain v1, nilai *P-value* memiliki nilai yang lebih dari atau sama dengan 0,01. Sehingga dapat disimpulkan bahwa *sequence keystream* tersebut random.

6.2.5.3 Grain 128

Tabel 6.15 Hasil *Binary Matrix Rank Test* Grain 128

No	Key	IV	F_M	F_{M-1}	N	X^2_{obs}	*P-value*
1.	0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000	16	18	4	3,2402	0,3839
2.	8000 0000 0000 0000 0000 0000 0000 0000	8000 0000 0000 0000 0000 0000 0000	14	16	8	4,1296	0,2468
3.	C000 0000 0000 0000 0000 0000 0000 0000	C000 0000 0000 0000 0000 0000 0000	13	25	0	5,8748	0,0872
4.	E000 0000 0000 0000 0000 0000 0000 0000	E000 0000 0000 0000 0000 0000 0000	14	21	3	1,7247	0,6917
5.	F000 0000 0000 0000 0000 0000 0000 0000	F000 0000 0000 0000 0000 0000 0000	13	21	4	0,6433	0,9083
6.	F800 0000 0000 0000 0000 0000 0000 0000	F800 0000 0000 0000 0000 0000 0000	16	18	4	3,2402	0,3839
7.	FC00 0000 0000 0000 0000 0000 0000 0000	FC00 0000 0000 0000 0000 0000 0000	18	14	6	7,5442	0,0269
8.	FE00 0000 0000 0000 0000 0000 0000 0000	FE00 0000 0000 0000 0000 0000 0000	12	23	3	0,8435	0,8725
9.	FF00 0000 0000 0000 0000 0000 0000 0000	FF00 0000 0000 0000 0000 0000 0000	12	21	5	0,1380	0,9838
10.	FF80 0000 0000 0000 0000 0000 0000 0000	FF80 0000 0000 0000 0000 0000 0000	15	18	5	2,1882	0,5917

Untuk *binary matrix rank test*, dari Tabel 6.15 dapat dilihat bahwa hasil pengujian Grain 128, nilai *P-value* memiliki nilai yang lebih dari atau sama dengan 0,01. Sehingga dapat disimpulkan bahwa *sequence keystream* tersebut random.

6.3 Analisis Hasil Pengujian

Dari pengujian validitas *keystream* didapatkan bahwa *stream cipher cryptography processor* algoritma Grain ini telah berjalan sesuai dengan algoritma yang dibuat dalam tahap implementasi. Dan *keystream* yang dihasilkan telah sesuai dengan referensi *test vector* Grain yang diberikan oleh referensi [HJM-05] dan [HJM-06]. Hasil pengujian validitas *keystream* menunjukkan bahwa *stream cipher cryptography processor* algoritma Grain tersebut telah memenuhi semua kebutuhan yang dirancang pada tahap perancangan.

Dari hasil *randomness testing* dapat dilihat bahwa *keystream* telah diuji dengan menggunakan 5 buah test standar dari *National Institute of Standard and Technology* (NIST) yaitu: *Frequency Test, Frequency Test within a Block, Runs Test, The Longest Run in a Block dan Binary Matrix Rank Test.* Nilai *P-value*, sebagai nilai acuan untuk menentukan tingkat keacakan bit harus memenuhi persyaratan NIST, yaitu harus lebih dari atau sama dengan 0,01. Dari hasil pengujian pada Tabel 6.1 sampai 6.15, nilai *P-value* setiap test telah memenuhi persyaratan NIST yaitu lebih dari atau sama dengan 0,01. Sehingga dapat disimpulkan bahwa *keystream* yang dihasilkan oleh *stream cipher cryptography processor* algoritma Grain telah memenuhi syarat tingkat keacakan bit.

BAB VII
PENUTUP

7.1 Kesimpulan

Dari hasil pengujian dan analisis hasil penelitian yang telah dilakukan dapat disimpulkan bahwa :

1. Implementasi *stream cipher cryptography processor* ini membutuhkan 5.193 *logic gates* dengan *throughput* 0,14 Gbps dan maksimum frekuensi 138,778 MHz pada *device* Xilinx *Spartan3E XC3S500 E* dengan *package FG320*. *Stream cipher cryptography processor* algoritma Grain ini telah siap digunakan untuk aplikasi lain yang membutuhkan keamanan data.
2. *Keystream* yang dihasilkan s*tream cipher cryptography processor* algoritma Grain ini telah teruji kebenarannya dengan menggunakan *test vector* Grain. *Keystream* yang dihasilkan oleh *stream cipher cryptography processor* algoritma Grain ini telah teruji tingkat keacakannnya dengan menggunakan test yang disediakan oleh *National Institute of Standard and Technology* (NIST), yaitu: *Frequency Test, Frequency Test within a Block, Runs Test, The Longest Run in a Block* dan *Binary Matrix Test*, dengan nilai *P-value* untuk masing-masing test telah memenuhi standar keacakan yaitu lebih dari atau sama dengan 0,01.

7.2 Saran

Saran yang dapat diberikan untuk pengembangan desain dan implementasi *stream cipher cryptography processor* algoritma Grain ini adalah dari hasil penelitian ini yang berupa komponen *Cryptography Processor*, dapat digunakan sebagai dasar penelitian lebih lanjut untuk digabungkan dengan komponen lain seperti: *Multi-Purpose Processor, Memory* dan *I/O Devices* untuk membentuk sistem yang lengkap yang sekarang berkembang dengan pesat yang disebut dengan *System-on-Chip* (SoC).

DAFTAR PUSTAKA

[Ash-90] Ashenden, P.J. 1990. *The VHDL Cookbook*. First Edition. Dept. Computer Science, University of Adelaide, South Australia.

[Bir-05] Biryukov, A. 2005. *Block Cipher and Stream Cipher : The State of the Art.* Katholieke Universiteit Leuven, Dept. ESAT/SCD-COSIC, Belgium.

[Ecr- 05] ECRYPT Network of Excellence eSTREAM – Update 1. September 2005.

[EJ-02] Ekdahl, P., Johansson T. 2002. *Another Attack on A5/1*. IEEE Transactions on Information Theory.

[Gar-05] Garret, Paul. 2005. *Abstract Algebra : Lectures and Worked Examples for Graduate Course*. University of Minnesota.

[GL-05] Gurkaynak, F. K. and Luethi, P. 2005. *Recommendations for Hardware Evaluation of Cryptographic Algorithms.* Integrated System Laboratory ETH Zurich, Switzerland.

[Han-00] Hankerson, D.R. et al. 2000. *Coding Theory and Cryptography, The Essentials*. Marcell Dekker, Inc

[HJM-05] Hell, M. Johansson, T. and Meier, W. 2005. *Grain – Stream Cipher for Constrained Environments*. Dept. of Information and Technology, Lund University, Sweden.

[HJM-06] Hell, M. Johansson, T. Maximov, A. and Meier, W. 2006. *A Stream Cipher Proposal: Grain 128*. Dept. of Information and Technology, Lund University, Sweden.

[Kea-02] Kean, Tom. 2002. *Cryptographic Rights Management of FPGA Intellectual Property Cores*. Algotronix Ltd. Edinburgh EH8 8YB United Kingdom. Security *FPGA 2002*.

[KLP-04] Kumar, S. Lemke, K. and Paar, C. 2004. *Some Thoughts about Implementation Properties of Stream Ciphers.* Horst Gortz Institute for IT Security, Ruhr Universitat Bochum, Germany.

[MOV-97] Menezes, Oorschot, v. P. and Vanstone, S. 1997. *Handbook of Applied Cryptography*. Boca Raton, CRC Press.

[NIS-04] National Institute of Standard and Technology (NIST). 2004. *Common Criteria for Information Technology Security Evaluation* (version 2.2) Revision 256.

[NIS-01] National Institute of Standard and Technology (NIST). 2001. *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications*. NIST Special Publication 800-22 (with revision dated May 15, 2001).

[Nyb-04] Nyberg, Kaysa. 2004. *Cryptographic Algorithms for UMTS*. Nokia Research Centre. European Congress on Computational Methods in Applied Sciences and Engineering 2004.

[Osw-05] Oswald, E. et al. 2005. *State of the Art in Hardware Architectur*e. ECRYPT European Network of Excellence in Cryptology, Information Socienty Technologies.

[Per-94] Perry, D.L. 1994. *VHDL*. McGraw Hill Book Co. International Edition.

[Ros-05] Rosen, KH. 2005. *Number Theory and Its Application*. AT&T Laboratories. Pearon Addition Wesley 2005.

[RRS-03] Ravi, S., Anand Raghunathan and Srimat Chakradhar. 2003. *Embedding Security in Wireless Embedded Systems*. IEEE Proceedings of the 16th International Conference on VLSI Design (VLSI'03)

[Rue-92] Rueppel, R. A. 1992. *Stream Cipher*. In G. J. Simmons editors. Contemporary Cryptology: The Science of Information Integrity. IEEE Press 1992.

[Rus-96] Rushanan, J. J. 1996. *A Tutorial on Finite Field and Binary m-Sequences.* The Mitre Corporation.

[Tur-05] Turan, M. S. 2005. *Statistical Analysis of Synchronous Stream Cipher*. Institute of Applied Mathematic, Middle East Technical University, Turkey.

[WP-04] Wollinger, Thomas and Christof Paar. 2004. *How Secure Are FPGAs in Cryptographic Applications?*. COSY Horst Gortz Institute for IT Security. Ruhr-Universitat Bochum, Germany.

[VSC-99] Visible System Co. 1999. *Visible Analyst Tutorial*. Visible System Corporation.

[Xil-06a] Xilinx Inc. 2006. *Programmable Logic Design. Quick Start Handbook*. Xilinx Inc.

[Xil-06b] Xilinx Inc. 2006. *Synthesis and Verification Design Guide*. Xilinx Inc.

[Xil-06c] Xilinx Inc. 2006. *Spartan-3E FPGA Family: Complete Data Sheet*. Xilinx Inc. DS312.

[Zen-04] Zenner, E. 2004. *Stream Cipher Criteria*. CRYPTICO A/S.