# Low Power Scalable Encryption For Wireless Systems

by

James R. Goodman

B. ASc. Electrical Engineering
University of Waterloo, Canada, 1994

Submitted to the Department of Electrical Engineering
and Computer Science in partial fulfillment of the
requirements for the degree of

Master of Science in Electrical Engineering and Computer Science

at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
August 28, 1996



Author ..... .....
Department of Electrical Engineering and Computer Science
August 28, 1996

Certified by .....
Anantha Chandrakasan
Analog Devices, Inc.Career Development Assistant Professor
Thesis Supervisor

Accepted by ..... .....
F. R. Morgenthaler
Chairman, Department Committee on Graduate Students

# Low Power Scalable Encryption For Wireless Systems

by

James R. Goodman

Submitted to the Department of Electrical Engineering and Computer Science on August 28, 1996, in partial fulfillment of the requirements for the degree of Master of Science in Electrical Engineering.

## Abstract

Secure transmission of information is critical for many wireless applications. Wireless transmission imposes constraints not found in typical wired systems such as low power consumption, tolerance to high bit error rates, and scalability. A variety of cryptographic algorithms have been analyzed for their suitability to low-power wireless systems, and low power techniques have been developed to reduce their power consumption. Several test chips were designed, fabricated, and tested to verify the results proposed within this work. in addition, a model for error propagation in block ciphers is proposed and verified through experimentation.

Thesis Supervisor: Anantha Chandrakasan
Title: Analog Devices, Inc. Career Development Assistant Professor

# Acknowledgements

First and foremost I'd like to say thank you to my advisor Anantha Chandrakasan who's sense of humour, patience, and constant drive (not to mention funding...) helped me finally get this damn thing done!

Thanks also to the members of "ananthagroup" for all of their help over the last two years: to Raj "Oh my god! You're on fire!" Amirtharajah for reminding me that everyone suffers here in Hell, to Vadim "Cadence is broken...." Gutnik for being a well of knowledge that I frequently drank from, to Abram "Where does he find time to be a father?!" Dancy for suffering with me through the TSMC experience, to Wendi "It's only 5 flights of stairs..." Rabiner for being the only one of us sane enough to stay away from chip design, to Duke "the XOR man" Xanthopolous for catching my mistakes when I was too tired to notice them myself, and Tom "I'm off to make real money." Barber for his constant friendship and transport to and from Analog Devices.

I'd also like to thank my roommates Phillip "It only started 20 minutes ago." Alvelda for the bad movies and great friendship, and Misha "The Anti-christ of Driving." Bolotski for being a Cadence god, and all around great friend. And let us not forget Tony, the long-lost brother that I never knew I had until I showed up for hockey that first month here at MIT. I owe you a lot my friend, you kept me going through the rougher spots of my stay.

Lastly I'd like to thank my family whom I owe everything: my sister Lisa and her husband Cory for pretending to recognize me when I came home every Christmas, my "Dad" Milt who is a gentleman in every sense of the word and more of a father to me than anyone else in this world, and my mother Cathy who I only wish were still here to share this with me. I miss you so very much Kate, there isn't a day that goes by that I don't think about you -- I love you very much.

*For Kate,*

# Table of Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

Two of the biggest trends in computing today are mobile computing and global networking. The popularity of the Internet is an example of the drive towards a "wired world" in which all computers exist within a global web that allows them to communicate and share information with other systems located around the world. At the same time the current trend in computing hardware is towards portable, battery-operated nomadic computing systems. Thus the average power consumption of the system must be minimized in order to maximize the battery lifetime and minimize the battery size/weight. The popularity of wireless networks is a direct result of these two trends as people strive to remain connected to the global web without having to be tied down to a wired link.

Unfortunately wireless networks are notorious for their susceptibility to tampering and eavesdropping. In a wired network, information is transmitted within protected physical links (wires), offering the user some measure of security. Wireless networks have no such protection and it is a simple manner to eavesdrop or interfere with a transmission (e.g., the top link from Figure 1-1). In the cellular phone network, for example, scanners can be purchased that allow an attacker to steal cellular phone identification codes that are then used to defraud the cellular service provider. Estimates place annual losses due to fraud for cellular phone service providers and their customers at over $500 million ([1], [2]). The need for secure wireless transmission becomes even more apparent when one considers the gradual migration of conventional commerce, electronic banking, and other sensitive applications to the Internet; a trend that will continue to wireless networks.

These wireless security schemes will require low power hardware encryption modules as building blocks upon which to build safe and reliable protocols to provide end users with a secure wireless link (e.g. the bottom link of Figure 1-1).

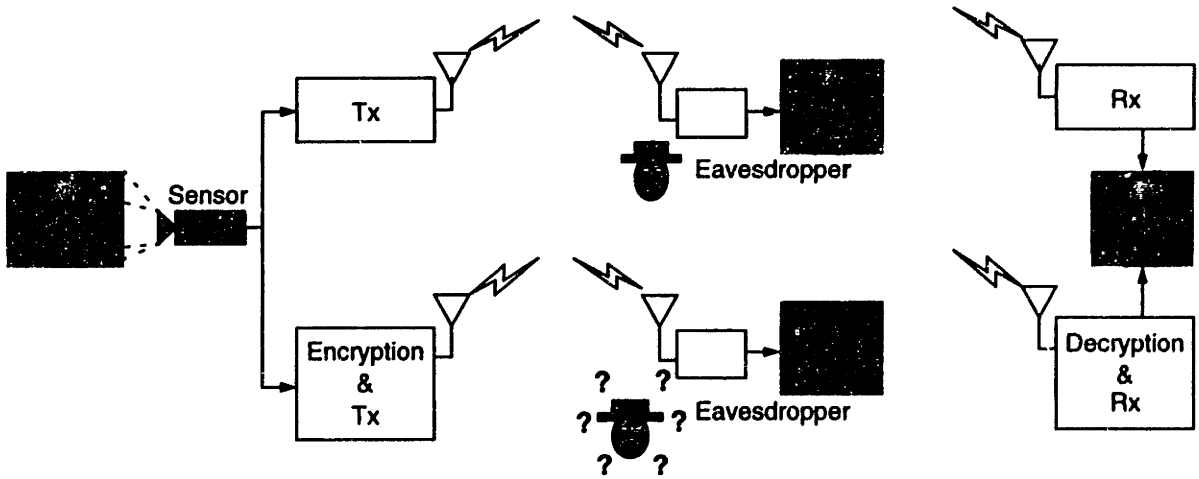



**Figure 1-1:** Wireless Network Encryption/Decryption Example

## 1.1 Low Power Design Basics

In modern integrated static CMOS circuits, the power consumption is dominated by the dynamic switching component that results from the charging/discharging of parasitic load capacitances. This power consumption can be modeled by the following expression:

$$P_{switching} = \alpha \cdot C \cdot V_{dd}^{2} \cdot f \tag{1-1}$$

where $C$ is the physical capacitance, $V_{dd}$ is the supply voltage and $f$ is the clock frequency. $\alpha$ represents the activity factor of the circuit which is the probability that $C$ will be charged on any given cycle. This models the fact that most capacitances are not being charged on every given cycle.

To minimize the power consumption of the circuit, the designer should minimize the variables of EQ 1-1. Techniques for minimizing the switched capacitance and supply voltage are described below.

### Minimizing the Switched Capacitance

Minimization of the switched capacitance can be done at several levels of the design hierarchy [3]. At the highest level, the designer can perform algorithmic optimizations that minimize the number of operations, as well as select a specific data representation that minimizes the bit switching activity (e.g., using sign-magnitude vs. 2's complement encoding of operands in a multiplier). At the circuit/logic level, the designer can optimize

the logic implementation and transistor sizing, as well as utilize power-down techniques to minimize the switched capacitance within the circuit.

**Minimizing the Supply Voltage -- Conventional Supply Scaling**

The switching component of the power consumption of a digital CMOS circuit scales with the square of the supply voltage. By minimizing $V_{dd}$ the designer can quadratically reduce the power dissipation of the circuit. Unfortunately propagation delays in CMOS integrated circuits increase as the supply voltage is reduced according to the first order model:

$$T_{delay} = \frac{k \cdot V_{dd}}{(V_{dd} - V_t)^2} \tag{1-2}$$

where $k$ is a process dependent constant, $V_{dd}$ is the supply voltage, and $V_t$ is the MOSFET threshold voltage. Figure 1-2 shows the normalized delay vs. supply voltage characteristic for a 0.6 µm process. This dependence limits the amount that the supply voltage can be reduced as the circuit must satisfy its throughput requirements. Hence the supply voltage is reduced until the critical path of the circuit is the same as the clock period required for the given throughput and architecture.

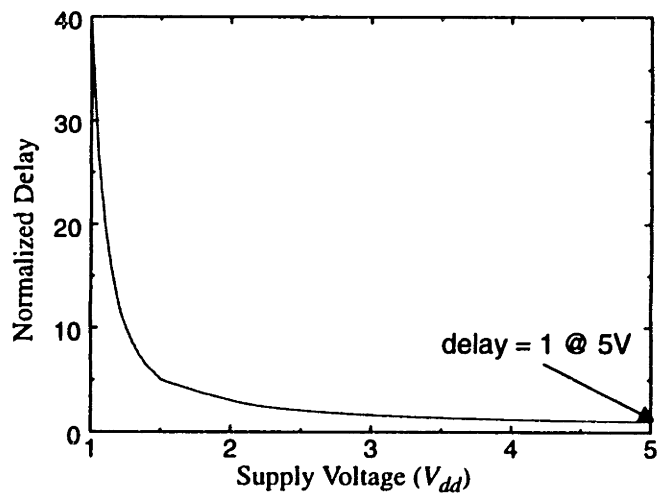



**Figure 1-2:** Normalized Delay vs. Supply Voltage

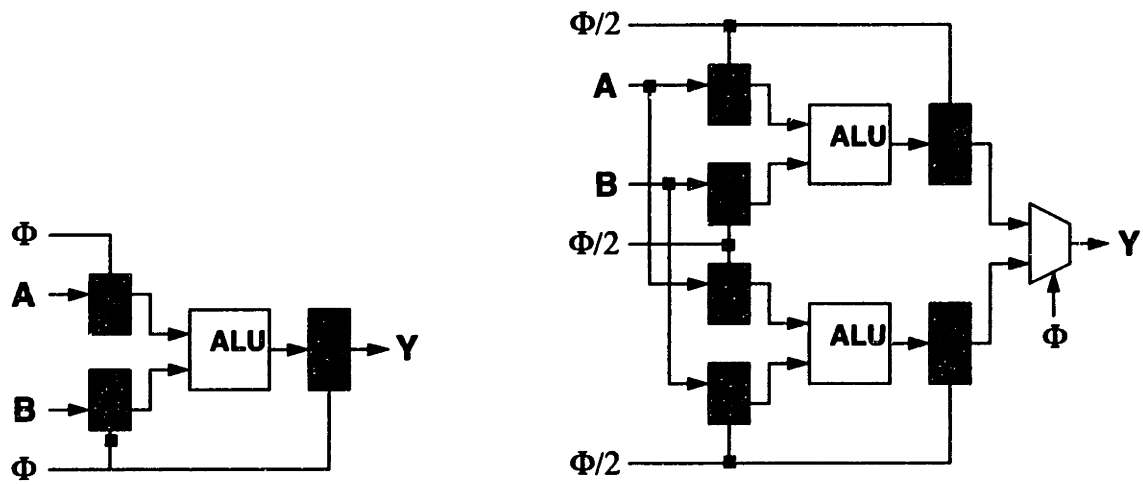
**Minimizing the Supply Voltage -- Parallelizing and Pipelining the Computation**

One technique for reducing the supply voltage involves parallelizing the computation to increase the effective cycle time of the circuitry for a fixed throughput. By increasing the

effective cycle time, the circuit delays can be increased and hence the supply voltage can be reduced. The cost of parallelizing the computation is the area overhead in creating parallel implementations of the circuitry and the increase in the switched capacitance due to the additional multiplexing and routing required. However the additional reduction in supply voltage will result in lower overall power consumption. As an example, consider the 2-way parallelization of an ALU (Figure 1-3). The parallel implementation datapath features twice the switched capacitance of the original, half the switching frequency and 0.58 times the supply voltage. The routing/muxing overhead is equivalent to 0.255 times the original switched capacitance and operates at the original switching frequency and 0.58 times the supply voltage. The overall normalized power consumption of the parallel implementation is thus:

$$P_2 = 2C \cdot (0.58V)^2 \cdot \frac{f}{2} + 0.255C \cdot (0.58V)^2 \cdot f = 0.422 \cdot CV^2 f = 0.422P,$$

which consumes 2.37 times less power than the original implementation [3].




$P = C \cdot V_{dd}^2 \cdot f$

$P_2 = (2C) \cdot (0.58V_{dd})^2 \cdot (0.5f) + C_{overhead} \cdot (0.58V_{dd})^2 \cdot f = 0.422\ P$

(a) Original Circuit

(b) 2-Way Parallelized Circuit

**Figure 1-3:** Parallelization Example: an ALU

In the general case, the power reduction factor for an n-way parallelization can be approximated using the following model assuming that the total switched capacitance scales with n plus some correction factor that is a function of the parallelism, $\gamma(n)$. If

delays scale according to EQ 1-2, then the supply voltage ($V_{dd}$) can be calculated from the delay ($T_d$) using the expression:

$$V_{dd}(T_d) = \left(\frac{1}{2} + \frac{V_t T_d}{k} + \sqrt{\frac{V_t T_d}{k} + \frac{1}{4}}\right) \cdot \frac{k}{T_d} \tag{1-3}$$

If we ignore $V_t$ (i.e., $V_t = 0$), the supply voltage scales inversely with the delay of the circuit. Thus for an n-way parallelized circuit, the supply voltage can be reduced by a factor of $n$ (as delays increase by a factor of $n$). Hence the power consumption of the circuit is reduced by a factor of:

$$\text{reduction factor} = \frac{P_1}{P_n} = \frac{C \cdot V^2 \cdot f}{nC(1+\gamma(n)) \cdot \left(\frac{V}{n}\right)^2 \cdot \left(\frac{f}{n}\right)} = \frac{n^2}{1+\gamma(n)}$$

Pipelining can be utilized in a similar manner to reduce the supply voltage by partitioning the calculation into $n$ discrete steps, each of which needs to be completed in the original clock period. Thus delays can be increased by a factor of $n$ and hence the supply voltage can be decreased by a factor of $n$. The cost of pipelining is the additional pipeline registers that must be added to the circuit.

## 1.2 Cryptography For Wireless Systems and Thesis Scope

There are three basic research areas that need to be addressed in the design of a security system for a wireless communications channel: authentication, key exchange, and encryption/decryption. In all of the explanations presented below it is assumed that two parties (i.e., sender and receiver) wish to communicate in the presence of an external observer (i.e., the attacker), who is attempting to eavesdrop, over an insecure channel using digital transmission techniques.

### Authentication

Authentication is the means by which the sender and receiver verify their identities to one another at the beginning of a communication. This is particularly difficult in digital communications systems as each party is only able to see easily-forged binary messages at the

output of the communication channel. The parties must therefore utilize some form of protocol which guarantees that a party can demonstrate irrefutable proof of identity even in the presence of tampering. Typically this is done by requesting that the other party demonstrate knowledge of a piece of information that only the intended recipient could know and then verifying that their reply is valid.

**Key Exchange**

Usually the sender and receiver need to share a piece of secret information called the key for the purposes of encrypting and decrypting the data traffic between them. In some of these applications both parties are assumed to already have a copy of the key. In others the key must be decided upon and then shared over an insecure link. Hence there must be some means by which the sender and receiver can agree upon and exchange the key in the absence of any shared information and a secure link. Algorithms and protocols exist by which a key can be generated and exchanged. These are for the most part very computationally intensive and unsuitable for a low-power wireless application.

**Encryption/Decryption**

Once the keys have been established the sender and receiver now need to encode and decode the data that is to be transmitted over the channel. The encryption algorithms must be chosen such that the amount of work required for an attacker to decrypt the data is prohibitively high. There is a wealth of good encryption/decryption algorithms available to the wireless systems designer. However the complexity and security of these algorithms vary widely. Care must be taken to select an algorithm that provides a satisfactory level of security at an acceptable level of complexity. A trade-off that is made all the more critical by the low-power requirements of a typical mobile wireless system.

While all three of the above areas are equally important, scope limitations allow only the encryption/decryption aspect to be studied here. Authentication and key exchange will be considered as future work.

## 1.3 Driver Application

The research described within this body of work is driven by the Ultra Low Power Wireless Sensor Project at MIT. The goal of this project is to design, fabricate, and characterize

a wireless image sensor (Figure 1-4) capable of operating over a wide range of data rates (e.g., 1 bps - 1 Mbps). The sensor features an imager, reconfigurable A/D converter, DSP core (which includes video compression and an encryption module) and a transmitter. The sensor electronics are intended to be generic in the sense that the imager can be removed and another form of sensor input can then be utilized with the same programmable A/D / DSP / RF transmitter core. The core must therefore be capable of operating at a variety of data rates and resolutions, as outlined in Table 1-1. The system features a variable voltage power supply for dynamic adjustment of the supply voltage. The sensor features a low bandwidth return channel from the base station to the sensor for control information, such as transmitter power control and Automatic Repeat Requests (ARQ) acknowledgments for a low data rate sensor configuration (e.g., temperature sensing).

The design specifications of the sensor require its size to be no greater than 1 $in^3$, with a battery operated lifetime of 1000 hours of active operation and 10,000 hours of stand-by operation. The power budget of the sensor system is approximately 50 mW.

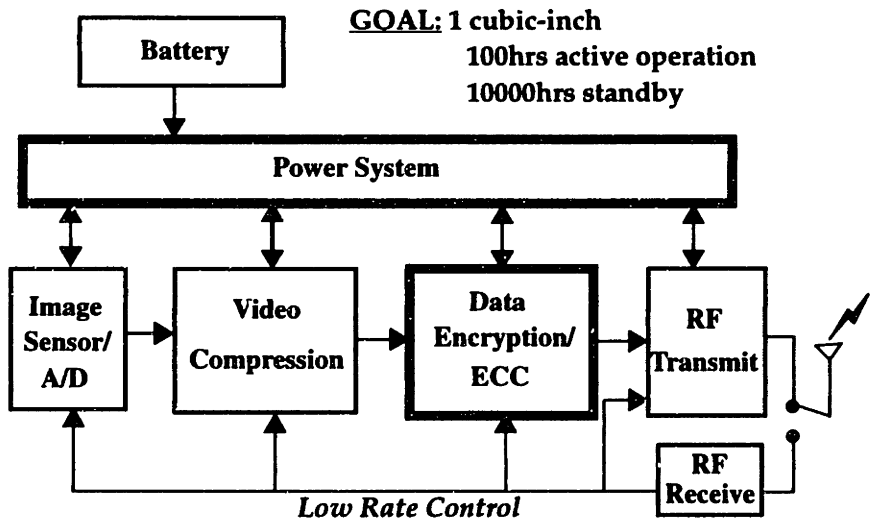



**Figure 1-4:** MIT Wireless Sensor Project

| Sensor Type | A/D Resolution (bits/sample) | Data Rate |
|---|---|---|
| Video (compressed) | 8 bits | ≤ 1Mbps |
| Audio (uncompressed) | 20 | ~ 800 kbps |
| Temperature | 20 | 20 bps |

**Table 1-1:** Various sensor applications and their data characteristics

# Chapter 2

# Introduction to Basic Cryptography

## 2.1 General Classification of Algorithms

Cryptographic algorithms involve performing various operations on a given input to produce an encoded output, based on the value of a secret control input called the key. The security of a cryptographic algorithm is dictated by the difficulty of deducing the input given the output and the encoding algorithm with the exception of the key. Thus the security of the algorithm rests solely in the secrecy of the key and the computational complexity of deducing its value. At the topmost level, cryptographic algorithms can be partitioned into two classes depending on the symmetry of the encryption/decryption algorithms and keys.

### 2.1.1 Asymmetric Algorithms

Asymmetric algorithms require no exchange of secret information between the sender and the receiver, as they do not utilize the same keys for encryption and decryption (i.e., the keys are asymmetric). The receiver generates the encryption and decryption keys and then publicly distributes the encryption key. This allows anyone to encode a message so that only the receiver can decode it. The idea of using an asymmetric algorithm was first proposed by Whitfield and Diffie [4] and has since come to be known as Public Key Cryptography.

The security of asymmetric algorithms relies on the difficulty of solving hard number-theoretic problems, such as computing discrete logarithms over a field (e.g., ElGamal [5]), or factoring large composite numbers (e.g., RSA [6]). As an example of the difficulty of such problems, consider that the estimated effort to factor a 512 bit composite number is $3 \times 10^4$ MIPS-years[1] [7]. Unfortunately, the complexity of these problems makes asymmetric algorithms much too computationally demanding for wireless applications where

1. a MIPS-year is the number of calculations performed by a machine capable of performing $10^6$ instructions per second, running non-stop for a full year

power budgets are strictly limited. They do however find applications in wireless networks for such operations as authentication and key exchange (e.g., [8] and [9]).

### 2.1.2 Symmetric Algorithms

Symmetric algorithms differ from asymmetric algorithms in that the sender and receiver must share a secret key that is used for both encryption and decryption. This implies that the keys can be exchanged through some secure channel, or can be installed within the encryption/decryption units during fabrication, thereby avoiding the whole key exchange process.

The security of a symmetric algorithm is based on the amount of computation that is required to determine the value of the secret key. This can be parameterized by defining an equivalent keylength that represents the number of values that must be checked in order to determine the secret key value. This set of possible key values is denoted as the keyspace of the algorithm.

Symmetric algorithms are in general much more efficient than asymmetric systems as the security of the algorithm is based on the secrecy of the key and not the computational complexity of the algorithm. Hence they are the algorithm of choice for power-limited wireless applications.

## 2.2 Block Cipher Algorithms

Block cipher algorithms are symmetric algorithms that operate on blocks of data, $n$ bits at a time, to generate an $m$ bit output that forms the encrypted message. Obviously $m$ must be greater than $n$ in order to have an invertible mapping that can be decrypted and, for the purposes of wireless communication networks where bandwidth is very limited, typically is restricted to $m = n$. As a result a block cipher can be thought of as a memoryless $n$ bit permutation of the inputs under the influence of the secret key.

Most practical block cipher implementations are constructed using a cascade of simple bitwise permutations and substitutions under the influence of a function of the secret key (Figure 2-1). Each stage contributes to the overall security of the cipher. This construction is known as a product cipher and serves as the basic structure of most popular block ciphers in use today (e.g., DES [10], IDEA [11], BLOWFISH [12], and RC5 [13]).

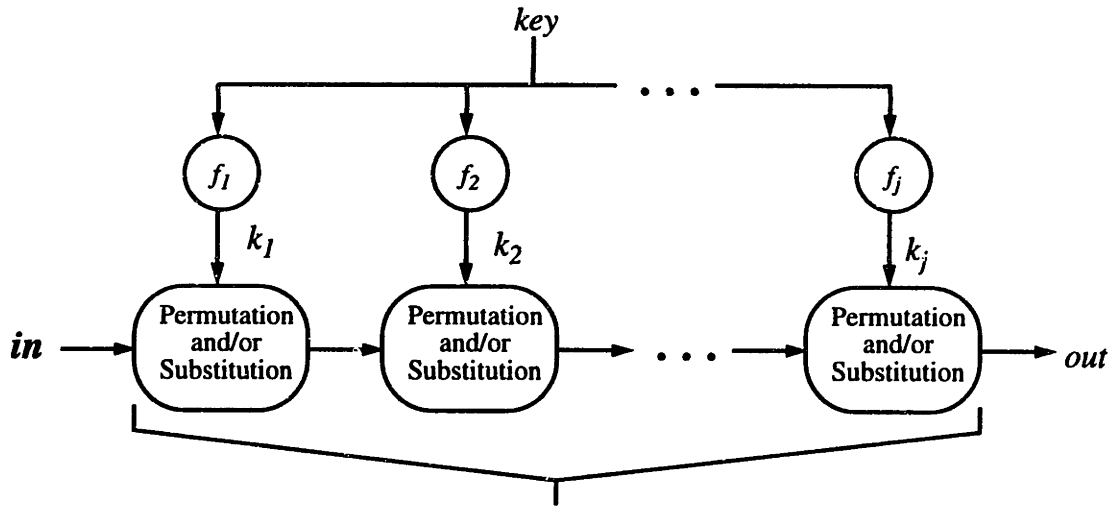


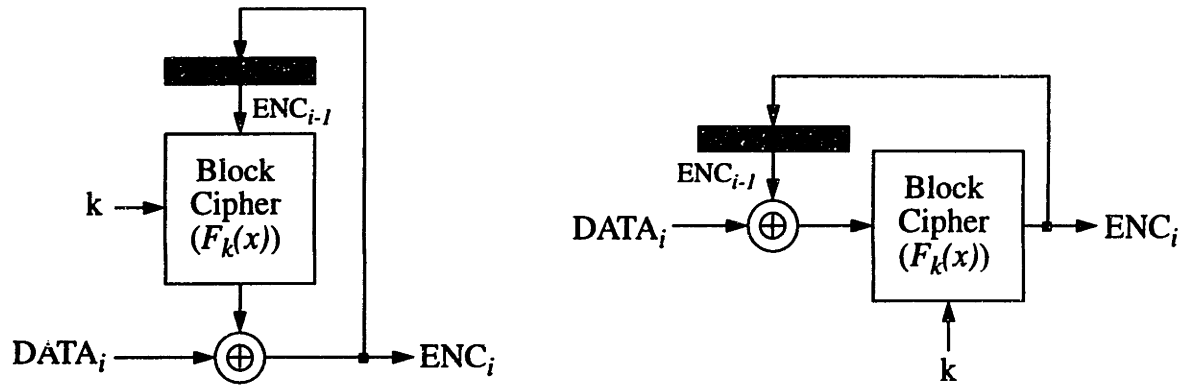

**Figure 2-1:** Product Cipher Construction

The idea can be extended to the cascading of multiple block ciphers to create a more secure cipher. The security of the overall system is not guaranteed to be $n$ times as difficult to crack if $n$ ciphers are cascaded together. The only guarantee is that the system will be at least as difficult to crack as the weakest algorithm used in the cascade [14].

The lack of memory in the block cipher results in the weakness that given the same input ($x$) and key ($k$), the block cipher will always output the same value, $y = F_k(x)$. This weakness can be overcome by utilizing the block cipher with external memory and feedback in one of the following operating modes (Figure 2-2):

- **Cipher Feedback (CFB):** data is XORed with the encryption of the previous block's output (i.e., $ENC_i = DATA_i \oplus F_k(ENC_{i-1})$).
- **Cipher Block Chaining (CBC):** data is first XORed with the previous block's output and the result is then encrypted (i.e., $ENC_i = F_k(DATA_i \oplus ENC_{i-1})$).

CFB can use a varying amount of feedback by only using $j$ bits of $ENC_i$ in the feedback loop. The contents of the feedback register are shifted $j$ bits to the left and the $j$ bits of $ENC_i$ are then loaded into the least significant $j$ bits of the feedback register.




(a) Cipher Feedback Mode (b) Cipher Block Chaining

**Figure 2-2:** Block Cipher Operating Modes

By introducing feedback into the encryption/decryption process, the system designer has also introduced error propagation. If so much as a single bit is in error at the decryption process input, it will decrypt to random values. These random values will then be used in the decryption of subsequent blocks, causing numerous errors to propagate across multiple blocks. In environments where bit errors are very likely (e.g., a wireless communications channel), this can lead to catastrophic error propagation. A detailed analysis of this problem is given in Section 2.5.1.

## 2.3 Stream Cipher Algorithms

Stream ciphers differ from block ciphers as they contain internal state that makes their output a time-dependent function, thereby avoiding the replay weakness that haunts block ciphers. In addition, a stream cipher operates on a data stream, typically one bit wide, rather than a block of data.

Stream ciphers are commonly modelled as a sequence generator that produces a time-varying pseudo-random sequence (i.e., the keystream) that is then XORed with the data stream to produce the encrypted data stream (Figure 2-3). Decryption is performed by using an identical keystream generator which is synchronized to the data stream. Its output is XORed with the encrypted data stream, removing the influence of the original keystream (i.e., $(k_i \oplus d_i) \oplus k_i = d_i$).

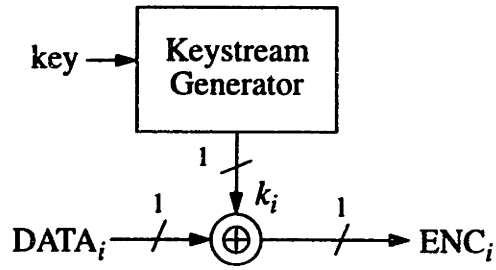



**Figure 2-3:** Stream Cipher - Keystream Generator Construction

Note that it is possible to construct a stream cipher using a block cipher by operating the block cipher in an Output Feedback Mode (OFB) configuration (Figure 2-4). In this configuration the block cipher is never used to directly encrypt the data, rather it is given an initial starting vector that it then repeatedly encrypts. On any iteration $m$ bits of $OUT_i$ are shifted into the least significant $m$ bits of the feedback register, whose contents are then encrypted to generate $OUT_{i+1}$.

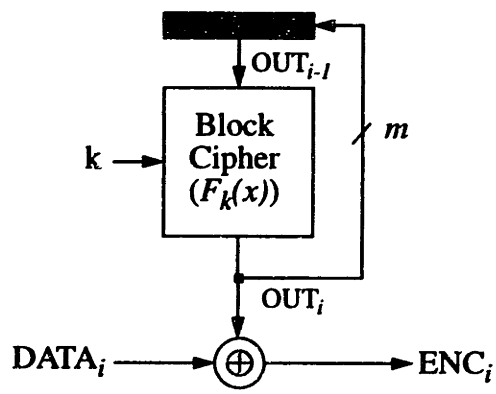



**Figure 2-4:** Output Feedback Mode

A very thorough discussion concerning the construction and analysis of stream ciphers is given by Rueppel in [15].

### 2.3.1 Design Criteria for Keystream Generators

Several stream cipher design criteria have been proposed that serve as necessary, but not sufficient conditions for obtaining a secure keystream generator [15]. These criteria are: sufficiently long period, sufficiently large linear complexity and uniformly distributed output sequence statistics.

The period of a keystream must be greater than the length of the data being encrypted. If not, the attacker can XOR two encrypted data bits to remove all influence of the key (i.e., $(a_i \oplus k_i) \oplus (a_{i+T} \oplus k_{i+T}) = a_i \oplus a_{i+T}$ as $k_i = k_{i+T}$). It is then a simple matter to decrypt the resulting data stream using simple statistical techniques[2].

The linear complexity (LC) of a keystream is a measure of the length of an equivalent LFSR that generates the keystream. The length of the equivalent LFSR and its feedback polynomial can be determined from 2·LC keystream bits using the Berlekamp and Massey algorithm [16]. Thus the linear complexity must be made sufficiently long that the attacker never sees enough keystream bits to mount such an attack.

In addition to a long linear complexity and period, the keystream must also exhibit signal statistics that appear indistinguishable from a uniform distribution. Any deviations could be utilized by an attacker to create a next bit predictor for the keystream that could then be used to crack the system.

While satisfying all of the aforementioned design criteria is a necessary condition for a secure keystream generator, it is by no means sufficient. In general, the designer must analyze any proposed algorithm to discover its weaknesses and quantify its strength to known attacks.

### 2.3.2 Linear Feedback Shift Register Based Stream Ciphers

Linear feedback shift registers (LFSRs) consist of a shift register whose input is computed using a linear recursion based on the current state of the shift register:

$$x_1(t+1) = \sum_{i=1}^{n} c_i \cdot x_i(t) \tag{2-1}$$

Alternatively, the feedback expression can be thought of as an $n$th degree feedback polynomial given by the expression:

$$f(x) = c_n x^n + c_{n-1} x^{n-1} + \ldots + c_1 x^1 + 1 = 0 \tag{2-2}$$

2. For detailed examples of statistical cryptanalysis based on the statistics of English see Section 1.2 of [36].

where both $c_i$ and $x$ are binary values. The above summations are performed modulo-2 and can be implemented using simple XORs, thus resulting in a very simple and efficient circuit (Figure 2-5).

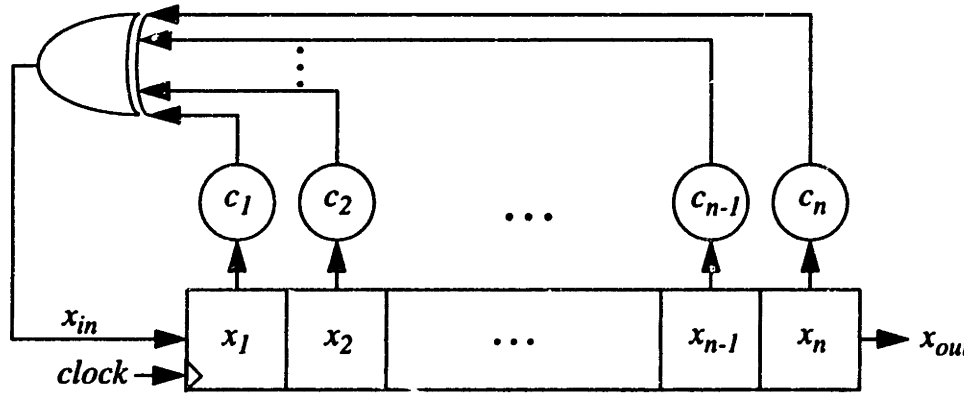



**Figure 2-5:** An $n$-bit LFSR

An $n$ bit LFSR can be in any of $2^n$ states and can cycle through any of $2^n$-1 states (the all 0 state will cause the LFSR to become stuck). Any LFSR that cycles through all $2^n$-1 non-zero states is said to be a maximal length LFSR and its output is called an m-sequence. The LFSR's feedback polynomial must be irreducible (i.e., there is no polynomial of degree $0 < k < n$ that divides $f(x)$) in order to generate an m-sequence. There are a total of $\phi(2^n\text{-}1)/n$ (see Section 4.1 for a definition of $\phi$) primitive polynomials for an $n$ bit LFSR. For large values of $n$, the number of primitive polynomials approaches $2^n$.

One interesting property of a maximal length LFSR is that its output exhibits a uniform distribution in terms of the number of 1's and 0's, making it appear to be an excellent candidate for use as a stream cipher keystream generator. Unfortunately LFSRs are completely insecure as any n-bit LFSR can be cracked after observing $2n$ output bits using the Berlekamp and Massey Algorithm. However, due to their simplicity and excellent output statistics, LFSRs are typically used as building blocks in more complicated stream cipher systems. Such systems utilize multiple LFSRs, that are clocked irregularly and whose outputs are combined using non-linear functions, to yield more cryptographically secure outputs.

## 2.4 Quadratic Residue Generator

The Quadratic Residue Generator (QRG) is a cryptographically-secure pseudo-random bit generator based on the difficulty of determining quadratic residues modulo a large composite modulus. The QRG was originally proposed by Blum, Blum, and Shub [17].

The QRG operates by performing repeated modular squarings: $x_i = x_{i-1}^2 \bmod N$ and extracting the least significant $log_2(log_2(x_i))$ bits from each result to generate the key-stream sequence. The modulus $N$ must be the product of two large prime numbers P and Q, where P and Q are both congruent to 3 modulo 4. The initial seed ($x_0$) must be a quadratic residue modulo $N$.

## 2.5 Comparison of Algorithms (Block Ciphers vs. Stream Ciphers)

Given that both stream ciphers and block ciphers are computationally efficient enough to be utilized in a low-power wireless application, it remains to be seen which is ultimately the most suitable. There are many considerations that must be studied, as outlined in the following sections.

### 2.5.1 High Bit Error Rates and Error Propagation

Block ciphers are designed so that inputs differing by so much as a single bit generate widely varying outputs. This property, while being highly desirable for security reasons, leads to the phenomenon of error propagation in block ciphers. Error propagation is the generation of multiple errors from a single error due to inter-bit dependencies and correlations in the data path. For wireless communication channels, one must be especially concerned with error propagation as BERs can be on the order of $10^{-2}$. When these large BERs are combined with signal processing algorithms, which are typically intolerant to even single bit errors (e.g., video compression algorithms), the need for forward Error Correction Coding (ECC) becomes obvious. However, ECC requires the introduction of redundancy into the data stream which consumes part of the transmission bandwidth. Thus it is desirable to minimize the amount of ECC that is required.

If raw pixel values are transmitted or the video compression algorithm is able to tolerate errors (e.g., pyramid vector quantization [18]), then error propagation analysis is extremely important as the increased BER translates into image degradation. Figure 2-6 shows the effects of error propagation for a stream cipher and 64-bit block cipher. As seen

from the figure, a single bit error will become a large burst of errors for the block cipher due to error propagation. Note that if the compression algorithm is intolerant to even single bit errors (e.g., huffman encoding [19]), error propagation is not an issue as it only involves the propagation of existing errors. Since an error has already occurred, the frame will be lost regardless of how many other errors are generated due to error propagation.

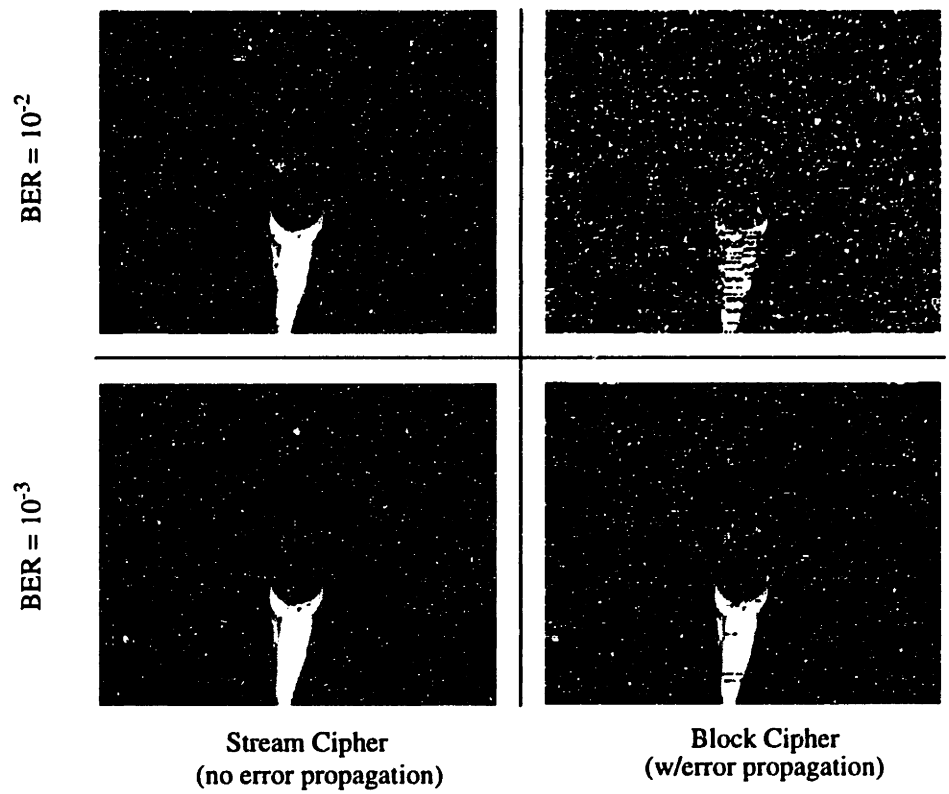



**Figure 2-6:** Effects of Error Propagation on an Uncompressed Video Image

Assuming that channel bit errors can be modeled as independent, identically distributed events with some probability $P_e$, then the channel BER for a block of length B bits can be expressed as:

$$BER = \frac{1}{B} \cdot \sum_{i=1}^{B} \left( iP_e^i (1-P_e)^{B-i} \binom{B}{i} \right) = P_e \tag{2-3}$$

Now assume that a block cipher is used to encrypt the data before it is transmitted over the channel. The block cipher operates on blocks of length B, and exhibits excellent diffusion characteristics such that even a single bit error will affect all other bits within the

block with probability 1/2. Hence a single bit error in the encrypted block will decrypt to a block with on average B/2 bit errors. Under these assumptions, the BER for a block of length B bits that is encrypted before being transmitted and then decrypted at the output of the channel (i.e., $BER_{encrypt}$) can be expressed as:

$$BER_{encrypt} = \frac{1}{B} \cdot \sum_{i=1}^{B} \left( \left( \frac{B}{2} \right) P_e^i (1 - P_e)^{B-i} \binom{B}{i} \right) \tag{2-4}$$

Figure 2-7 shows the resulting $BER_{encrypt}$, assuming a typical block length of 64 (e.g., IDEA and DES algorithms) for channel BER values ranging from $10^{-1}$ to $10^{-5}$.

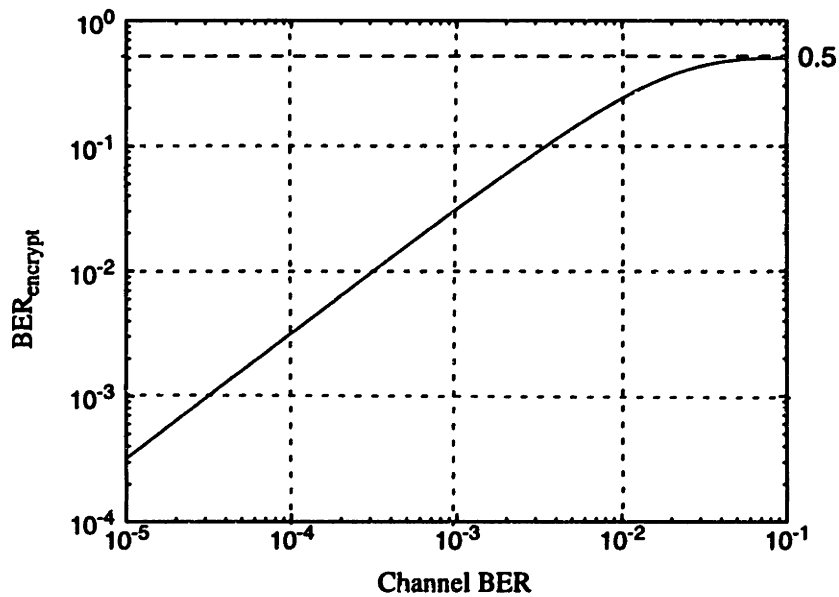



**Figure 2-7:** Theoretical BER for Block Size = 64 bits

Experiments were conducted using a 100-frame video sequence. The channel BER was varied from $10^{-2}$ to $10^{-4}$ and the 64-bit IDEA block cipher was used for encryption/ decryption. Figure 2-8 shows the multiplicative effects of error propagation for both the simulation and that predicted by EQ 2-4.

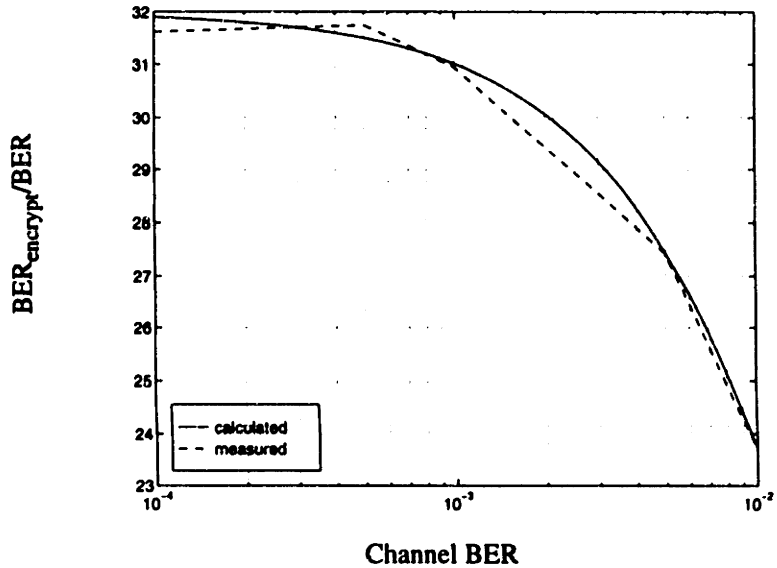



**Figure 2-8:** Error Propagation Ratio

### 2.5.2 Synchronization Effects

Wireless systems can experience deep fades due to obstructions in the communication channel. The result of these deep fades can be a loss in synchronization between the sender and receiver. In such cases the receiver will detect erasures in the transmitted data stream.

Both block and stream ciphers require constant synchronization between sender and receiver. When synchronization is lost in a block cipher system, the decryption block becomes unaligned resulting in random data (Figure 2-9). Similarly, when stream cipher keystream generators become unsynchronized the results are fatal as well. Recall that the data stream is XORed with the pseudo-random keystream to generate the encrypted data stream. Decryption requires that the encrypted data stream be XORed again with the synchronized keystream in order to remove the effects of the key (i.e., $A \oplus B \oplus A = B$). If the two generators ever become unsynchronized, the second XORing will not decrypt the data, rather it will re-encrypt it with the offset keystream (Figure 2-10).

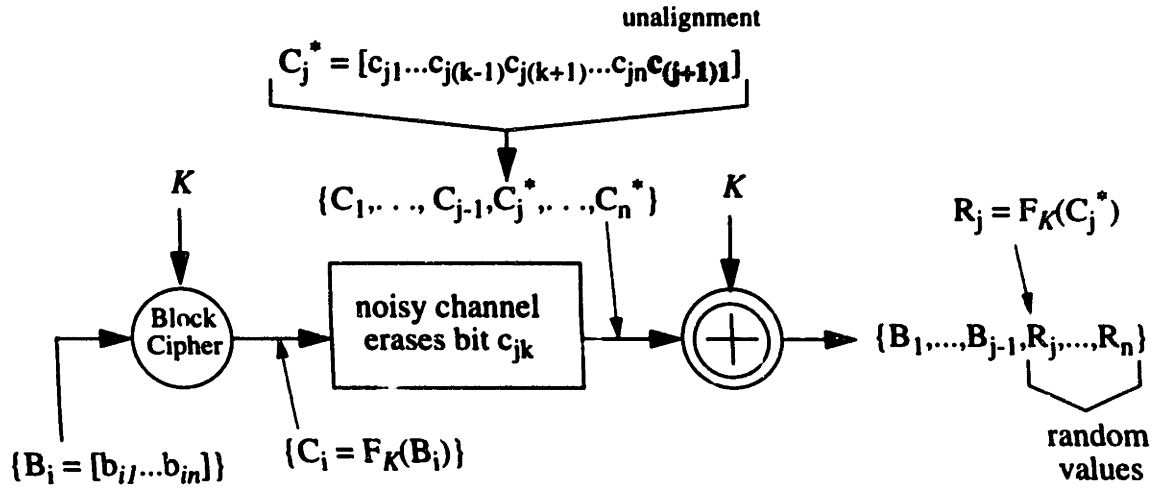



**Figure 2-9:** Effects of De-Synchronization on Block Ciphers

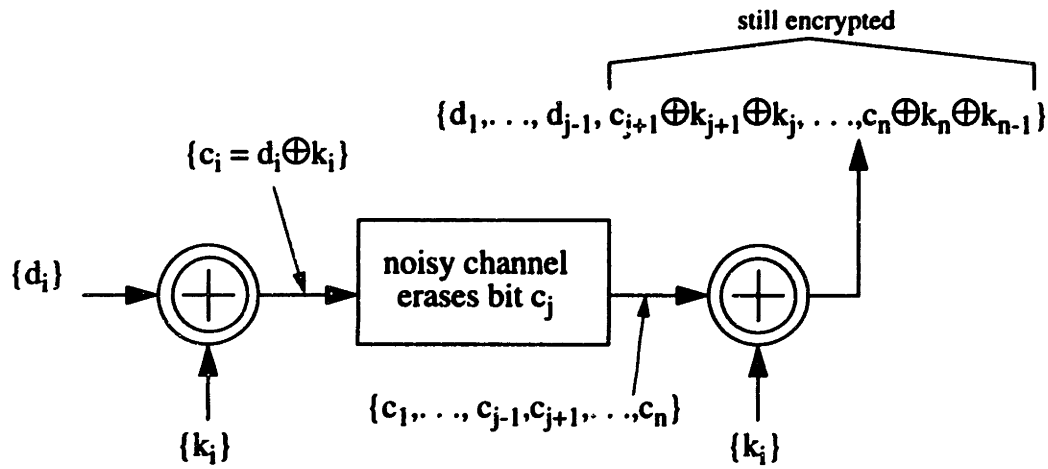



**Figure 2-10:** Effects of De-Synchronization on Stream Ciphers

One way to reduce the effects of synchronization loss is to force a periodic re-synchronization of the keystream generator with new seeds and feedback polynomials. If a sufficiently secure pseudo-random generator is used to generate the new seeds/polynomials, the attacker must start all over again in their attempt to crack the system. This makes the system very computationally expensive to crack and reduces the amount of information leaked for any given successful attack.

Another way to combat the effects of synchronization loss is to use a self-synchronizing cipher. A cipher is said to be self-synchronizing if the output depends on previous output values. Given a self-synchronizing system whose output is a function of the previous *n*

outputs, a missing output value will only affect the next $n$ output values. Hence, any correct continuous sequence of $n$ output values is sufficient to resynchronize the system. Compare this to a non-self-synchronizing system in which any one missing output value will cause all subsequent values to be incorrect until the system is re-initialized. The cost of using such a system is error propagation, as any incorrect output will affect $n$ other values. Given the high bit error rates of wireless systems (e.g., $10^{-2}$), this can prove to be prohibitively expensive in terms of the overhead of additional error correction coding.

In summary, both block ciphers and stream ciphers suffer from catastrophic error propagation in the presence of synchronization loss.

### 2.5.3 Buffering Requirements

Block ciphers require that their inputs be fixed-size $n$-bit blocks of data. If less than $n$ bits need to be encrypted then either the data can be buffered up until a full block is ready to be encrypted or the incomplete data block can be padded to create a full block. In applications where latency requirements don't allow for indefinite buffering delays, padding is the only option. However, padding reduces the efficiency of the algorithm, as the same amount of work is being performed to encrypt fewer bits. In an application where the data rate is widely varying, this inefficiency can result in significant additional power consumption.

On the other hand, stream ciphers require a minimal amount of buffering as they have an effective block size of a single bit. Hence, they are perfectly efficient in that no additional work needs to be performed.

### 2.5.4 Decoupling the Encryption Function

Block ciphers cannot be decoupled from the data stream as the outputs on any given cycle must be generated directly from the inputs (Figure 2-11). Stream ciphers, however, can be decoupled from the data stream since the encryption function involves XORing the data stream with the keystream, which can be precomputed and stored until it is required. This decoupling property allows stream ciphers to utilize additional power reduction techniques that are not possible with block cipher systems.

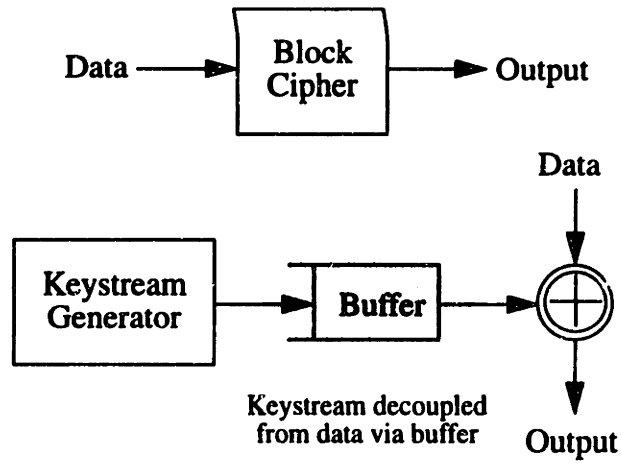



**Figure 2-11:** Comparison of Decoupling in Block and Stream Ciphers

## 2.6 Scalability

The scalability of a cryptographic algorithm can be defined as the measure of how the security of the cipher scales with some fundamental parameter associated with the algorithm. For LFSR-based ciphers, this parameter is typically the register lengths; while for the QRG it is the size of the modulus.

An LFSR-based stream cipher is cracked by reconstructing the contents and feedback polynomial of its constituent LFSRs. Knowing this information, an attacker can fully reconstruct the entire keystream and hence decrypt the data. As previously stated, an $n$-bit maximal-length LFSR can be in any of $2^n$-1 states and has any of $\phi(2^n\text{-}1)/n$ possible feedback polynomials, for a total of $(2^n\text{-}1)\phi(2^n\text{-}1)/n$ possible key values. However, not every key value needs to be tried as weaknesses in the algorithm allow the attacker to reduce the size of the keyspace to something significantly smaller. The size of this reduced keyspace determines the effective key length of the algorithm, which is the measure of the true difficulty to crack an LFSR-based stream cipher. Hence, the security of an LFSR-based stream cipher scales exponentially with the effective key size (i.e., security is $O\left(2^{n_{eff}}\right)$ where $n_{eff}$ is the effective key length of the algorithm). The relationship between the actual key size and the effective key size of an LFSR-based stream cipher varies from algorithm to algorithm and must be evaluated on a case-by-case basis.

The security of the QRG is based on the difficulty of factoring large integers and as such, its security varies with the difficulty of factoring its modulus. This in turn scales

with the size of the modulus. There are numerous algorithms for factoring large integers (e.g., the Number Field Sieve [20] and Quadratic Sieve [21]), the running time of which all have the general form $L(N,v,a+o(1)) = \exp((a+o(1))\cdot(\ln N)^{v}\cdot(\ln\ln N)^{(1-v)})$, where $a$ and $v$ are algorithm-specific constants (e.g., Table 2-1) and $N$ is the $n$-bit integer to be factored. The o(1) term is difficult to calculate and is eliminated from the analysis by approximating the time to factor $N$ given the known time to factor some integer $M$ using the expression:

$$T_N = T_M \cdot \frac{L(N, v, a)}{L(M, v, a)} \tag{2-5}$$

Practice has shown this to be a reasonable approximation [7]. The current best known algorithm for factoring large integers is the General Number Field Sieve (GNFS). A variant of the GNFS is the Special Number Field Sieve (SNFS) that can be used with integers of the form $(a^k \pm b)$ where $a$ and $b$ are small (typically $a = 2$, $b = 1$). Using the current version of the GNFS it is predicted that it will require $3\text{x}10^4$ MIPS-years to factor a 512 bit number. In general the difficulty of factoring is exponentially related to the length of the number being factored and hence the security of the QRG scales exponentially with the size of the modulus used (i.e., security is $O\left(e^{n^{v}(\ln(\ln 2)+\ln 2\cdot\log_2 n)^{1-v}(\ln 2)^{v}}\right)$ where $n$ is the length of the modulus). Figure 2-12 shows how the amount of computation scales with modulus length for the GNFS algorithm.

| Algorithm | $a$ | $v$ |
|---|---|---|
| General Number Field Sieve | $(64/9)^{1/3}$ | 1/3 |
| Special Number Field Sieve | $(32/9)^{1/3}$ | 1/3 |
| Quadratic Sieve | 1 | 1/2 |

**Table 2-1:** Algorithm Parameters for Several Factoring Algorithms

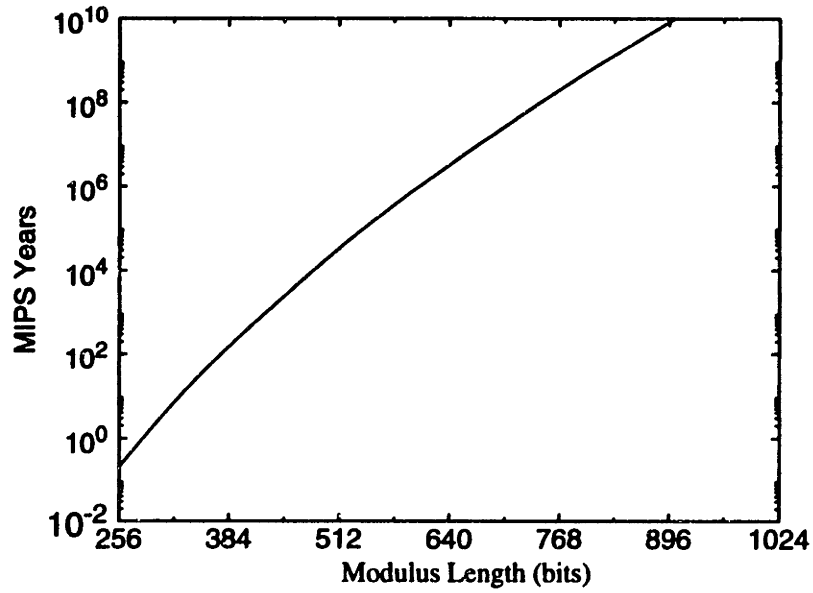



**Figure 2-12:** Amount of Computation Required for Factoring Using the GNFS

## 2.7 An Existing System - the Group Special Mobile (GSM) Standard

GSM is the third-generation cellular telecommunications standard currently being introduced in Europe and represents the most secure cellular telecommunications standard in existence today. The provision for radio link encryption/decryption is defined in the standard itself [22] and goes far beyond the primitive security standards of the Advanced Mobile Phone Standard (AMPS) system that is currently in place in North America. GSM utilizes three separate algorithms for performing authentication, temporary key generation and actual data encryption/decryption:

- **A3:** a 128-bit to 64-bit one-way hash function used for performing user authentication
- **A5:** a symmetric 64-bit key stream cipher used for data encryption/decryption
- **A8:** a 128-bit to 64-bit one-way hash function used for generating temporary session keys

The protocol relies on the existence of a unique secret subscriber ID key ($K_{ID}$) that is stored within a Subscriber Identification Module (*SIM*) [23] carried by the user and within the Authentication Center (*AUC*) [24] of the GSM network (Figure 2-13). Encryption is performed only over the wireless link. Once inside of the wired portion of the network it is transmitted in an unencrypted format.

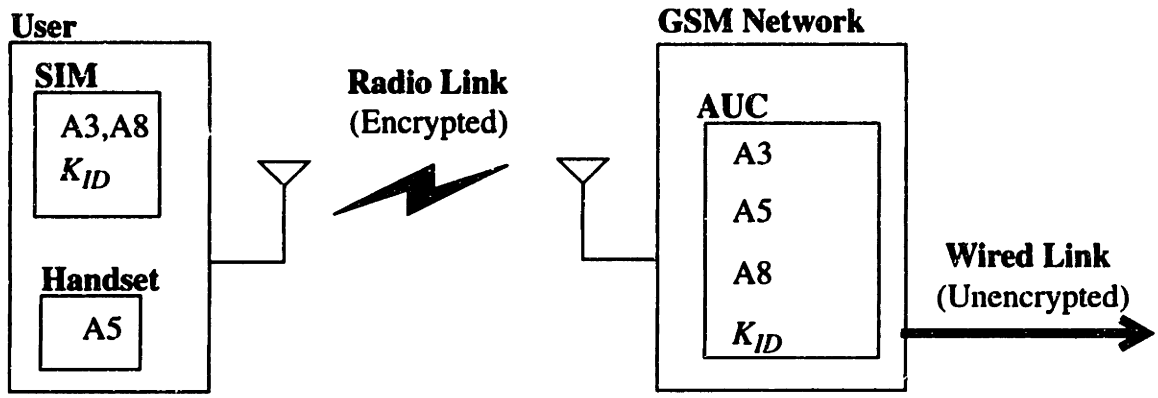



**Figure 2-13:** Distribution of Security Features in a GSM Network

At the start of a call, the network sends a 128 bit random value (*RAND*) to the user who in turn hashes this value under the influence of $K_{ID}$ to generate a 64 bit response (*RES* = A3[*RAND*,$K_{ID}$]) which it returns to the network. The network compares this value with the expected value (determined by applying A3 to its copy of *RAND* and $K_{ID}$) and if the two match, it concludes that the user is in fact who they claim to be (as only they would know the value of $K_{ID}$) and allows the call to continue. If encryption is required, the user generates a temporary session key ($K_S$) by applying A8 to *RAND* under the influence of $K_{ID}$. All data transfers are then encrypted/decrypted using A5 keyed with $K_S$. This protocol is shown in Figure 2-14.

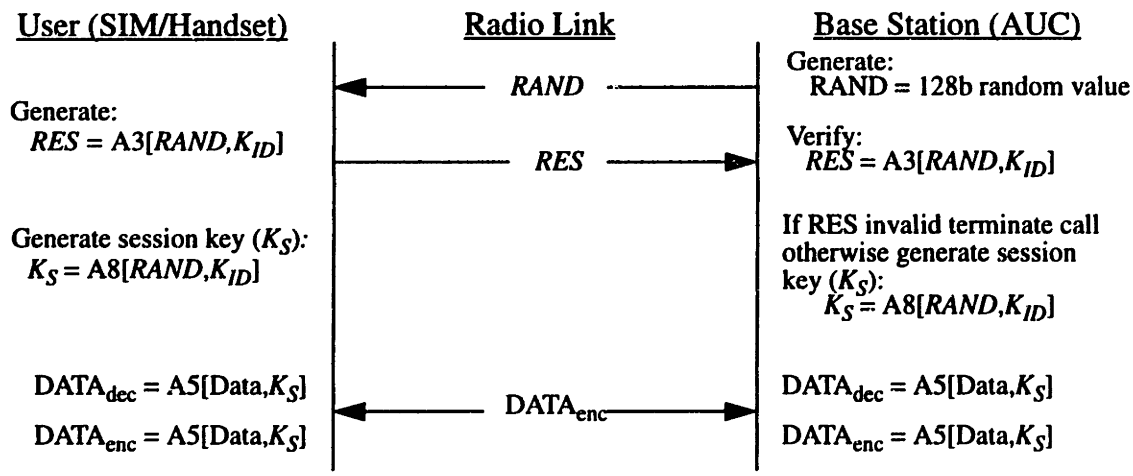



**Figure 2-14:** GSM Security Protocol

The GSM protocol demonstrates several features that are essential to a usable wi eless communication system and which should be considered by a wireless security systems designer:

- No secret information is ever allowed onto the unsecure radio link, it is always stored away inside of the SIM or AUC, both of which are assumed to be secure from tampering by an attacker.
- Data encryption keys are generated for each session, thereby forcing an attacker to start each attack from scratch.
- Users must authenticate themselves via a challenge-response protocol in which the network forces the user to generate a value that can only be calculated with knowledge of some secret information that is assumed to be known only by the intended receiver.
- There is no need for a complicated key exchange protocol as both the user and the network have copies of the secret key ($K_{ID}$) that is distributed with the SIM given to the user when they register with the network.

# Chapter 3

# LFSR-Based Stream Cipher Systems

The simplicity of LFSRs make LFSR-based stream ciphers a very attractive alternative for hardware implementations. Numerous LFSR-based stream cipher designs have been proposed and subsequently broken due to weaknesses in their construction (e.g., the Stop and Go Generator [25], the Geffe Generator [26], and the Jennings Generator [27]). Here several LFSR-based stream ciphers that appear to be secure, to the limits discussed within each description, are discussed. The power consumption of each algorithm is then estimated and various techniques for reducing the power consumption are described. Finally a test chip implementation of the algorithms is described and the test results are presented.

## 3.1 Shrinking Generator

The Shrinking Generator was first proposed by Coppersmith et. al. [28] as an efficient and reasonably secure (i.e., no obvious weaknesses) pseudorandom bit generator for use in stream cipher systems. The generator is based on the idea of a variable decimation of a pseudorandom sequence.

The Shrinking Generator utilizes a selection LFSR ($LFSR_{sel}$) to determine whether the output of the generating LFSR ($LFSR_{gen}$) is to be used within the keystream (Figure 3-1). If the output of $LFSR_{sel}$ is a "1" then the output of $LFSR_{gen}$ is added to the keystream, if not, then the output of $LFSR_{gen}$ is discarded. The output of the generator represents a varying decimation of $LFSR_{gen}$, as governed by $LFSR_{sel}$.

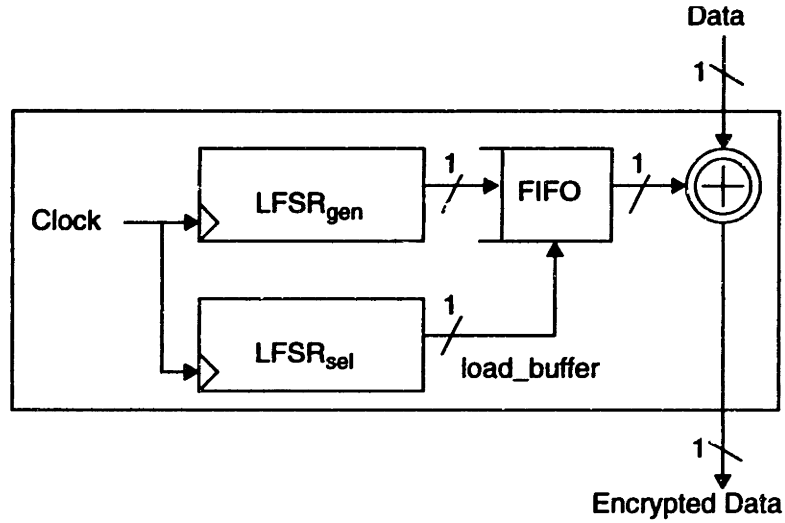



**Figure 3-1:** Shrinking Generator

Assuming that both $LFSR_{sel}$ and $LFSR_{gen}$ are maximal length and their periods are relatively prime (i.e., $\gcd(T_{sel},T_{gen}) = 1$), then the linear complexity and period of the Shrinking Generator are given by the expressions:

$$LC > |G| \cdot 2^{|S|-2}$$

$$T = \left(2^{|G|}-1\right) \cdot 2^{|S|-1}$$

where $|G|$ and $|S|$ are the lengths of $LFSR_{gen}$ and $LFSR_{sel}$ respectively. Assuming a data rate of 1 Mbps and choosing $|S| = 44$ and $|G| = 41$, the attacker will have to wait at least 10 years to obtain enough output values to mount an attack based on the linear complexity or period. The exponential dependance of both the linear complexity and period on the length of the selection register implies that $LFSR_{sel}$ should be as long as possible.

One obvious shortcoming of the Shrinking Generator is that it does not guarantee a keystream bit will be output on any given clock cycle. Hence the generator must utilize a higher clock frequency and an output buffer to ensure that there is always a keystream bit available. Kessler and Krawczyk proposed a model [29] for determining the required clock rate and buffer length in order to guarantee a certain miss probability and also proposed an empirical expression for estimating the miss probability given a certain buffer length ($B$) and clock rate multiplier ($\alpha$):

$$M(\alpha, B) \approx e^{-6.779 - 1.335\alpha B + 1.731\alpha^2 + 2.554B} \quad (3\text{-}1)$$

For large buffer sizes (e.g., 1024 bits) and modest clock rate increases (e.g., 2x), the miss probability can be effectively forced to zero ($M(2,1024) = 3x10^{-52}$).

Currently the best known attack on the Shrinking Generator reduces its effective key length to $|S|$ or $2|S|$ if the feedback polynomials are programmable. Another attack, based on Linear Cryptanalysis, was inferred by Golic [30]. Such an attack would require approximately $40(6.28 \cdot r/w)^w$ keystream bits to detect a weakness that could then be exploited, where $r$ is the length of $LFSR_{gen}$ and $w$ is one less than the number of non-zero feedback polynomial coefficients of $LFSR_{gen}$. Figure 3-2 show the infeasibility of such an attack assuming a keystream generator output rate of 1 Mbps. For a modestly sized $LFSR_{gen}$ (r = 32 bits) with only 10 non-zero feedback polynomial coefficients, it would require approximately 13.6 years to observe a sufficient amount of keystream bits to mount an attack.

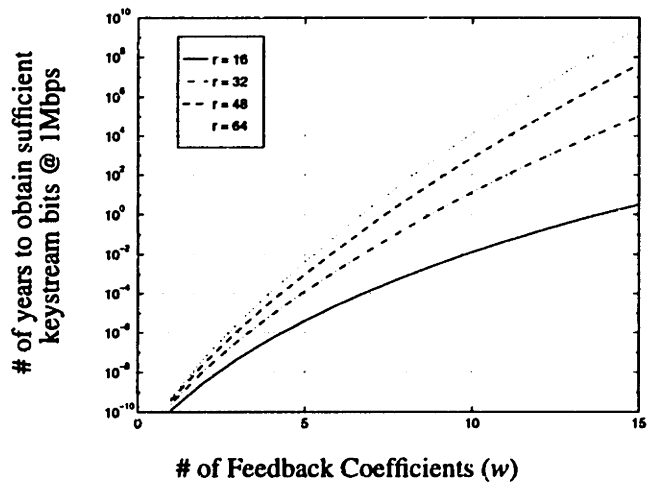



**Figure 3-2:** Time to Accumulate Keystream Bits for Linear Cryptanalysis of Shrinking Generator

## 3.2 Self-Shrinking Generator

The Self-Shrinking Generator [31] is a modification of the Shrinking Generator to reduce the hardware requirements by incorporating both $LFSR_{sel}$ and $LFSR_{gen}$ into one LFSR (Figure 3-3). On each clock cycle, the most significant bit of the LFSR is examined and if it is a "1" the second most significant bit of the LFSR is added to the keystream. The shift

register is then clocked twice and the whole process is repeated with the two new most significant bits.

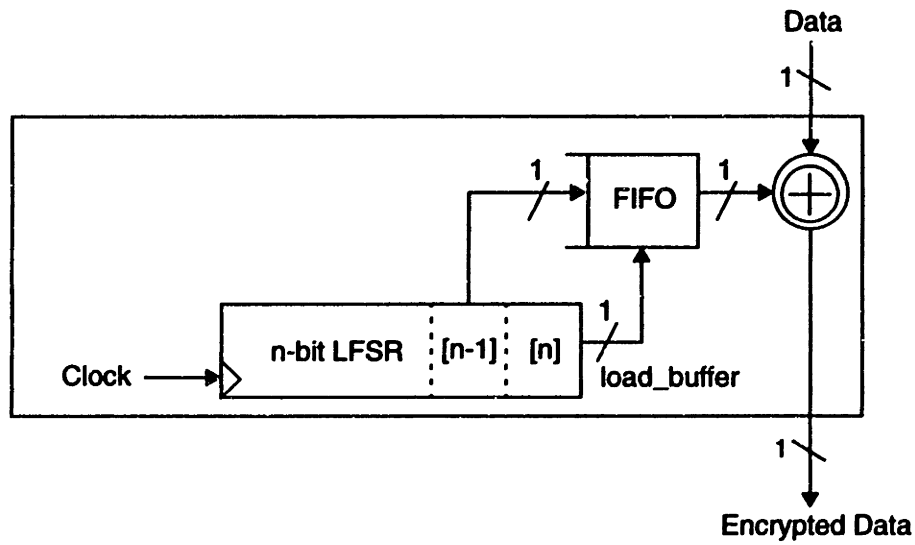



**Figure 3-3:** Self-Shrinking Generator

If the LFSR is a maximal length LFSR, the linear complexity and period of the Self-Shrinking Generator are given by the expressions:

$$T \geq 2^{\lfloor n/2 \rfloor}$$

$$LC > 2^{\lfloor n/2 \rfloor - 1}$$

where $n$ is the length of the LFSR. Meier and Staffelbach have conducted experiments [31] where they have discovered that for all $n$ such that $3 < n < 20$ the period is always maximal (i.e., $2^n - 1$) given a maximal length LFSR. In addition, the linear complexity was found to be very close to maximal (i.e., $2^{n-1} - 1$) for $n < 16$. Continuing the assumption of a data rate of 1 Mbps and choosing $n = 101$ bits, an attacker will require approximately 32 years to obtain enough output values to mount an attack using either the linear complexity or period of the Self-Shrinking Generator.

Early cryptanalytic results provided in [31] demonstrate that the Self-Shrinking Generator has an effective key length of $2^{0.375n}$ with a fixed feedback polynomial and $2^{0.875n}$ with a variable feedback polynomial. This compares favourably with the effective key

length of the Shrinking Generator which is approximately $2^{0.5(nsel + ngen)}$. However, it must be emphasized that these are preliminary results -- the algorithm is still too new to have had a thorough cryptanalytic analysis. Only time and scrutiny will tell just how secure the Self-Shrinking Generator really is.

One obvious drawback of the Self-Shrinking Generator is that the LFSR must be clocked at twice the rate of the Shrinking Generator, which in turn is operating at some multiple of the data rate. This higher clock rate will directly affect the power consumption of the algorithm.

## 3.3 Alternating-Stop-and-Go Generator

The Alternating Stop and Go Generator [32] utilizes two clock-controlled LFSRs whose outputs are XORed together to produce the keystream. The clock control is performed using a third LFSR whose output determines which of the two LFSRs is to be clocked on any given cycle (Figure 3-4).

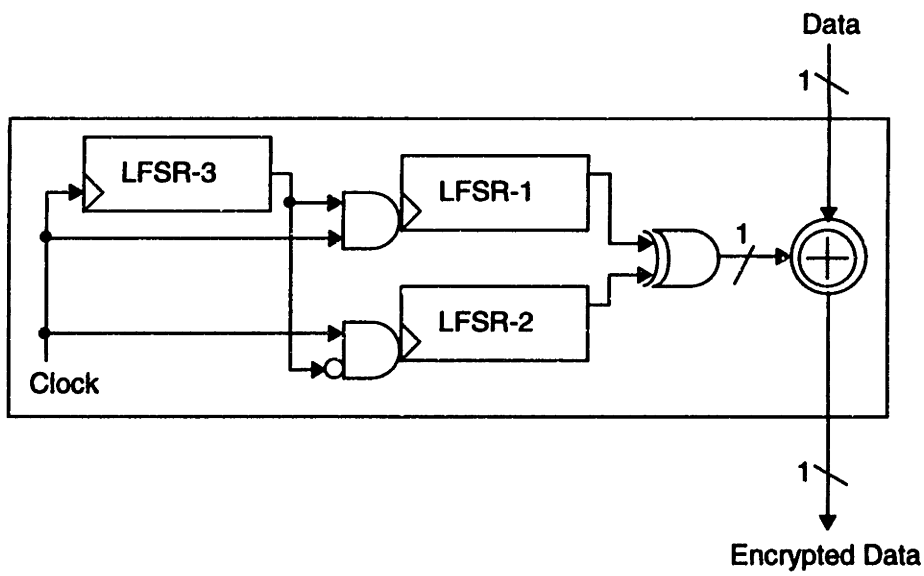



**Figure 3-4:** Alternating Stop & Go Generator

Assume that the $LFSR_1$, $LFSR_2$, and $LFSR_3$ are maximal length LFSRs of lengths $n_1$, $n_2$, and $n_3$ respectively and the periods of $LFSR_1$ and $LFSR_2$ are relatively prime. The period and linear complexity of the generator are then given by the expressions:

$$T_{ASG} = T_{LFSR3} \cdot T_{LFSR2} \cdot T_{LFSR1} \approx 2^{n_1+n_2+n_3}$$

$$LC > (n_1 + n_2) \cdot \left(2^{n_3} - 1\right)$$

With a 1 Mbps data rate and LFSRs of length 29, 31, and 47 bits respectively an attacker will require over 7 years to obtain enough output values to attack using either the linear complexity or period.

The best known attack on the Alternating Stop & Go Generator is a correlation attack on $LFSR_3$ [32] that effectively reduces the number of keys that must be searched to its third root (i.e., the length of $LFSR_3$), assuming that all LFSRs are of similar length. This attack can be overcome by choosing a sufficiently large value for $n_3$.

## 3.4 Proposed GSM Cipher (A5)

The proposed stream cipher algorithm (A5) used in the GSM cellular network standard is somewhat of a mystery and still officially remains a secret. However, several noted cryptographers have speculated as to its design and have proposed the construction described below ([33], [34]) based on information that they have gleaned from various sources. A5 utilizes three fixed-length LFSRs ($LFSR_1$, $LFSR_2$, and $LFSR_3$) with fixed feedback polynomial connections (Figure 3-5). The clock control function of the registers is derived by XORing the inverted carry-out bit from the summation of the three LFSRs' middle bits with that register's middle bit. This ensures that at least two of the three registers are clocked on any given cycle.

Since A5 is not yet officially released, the linear complexity and period of the cipher have yet to been quantified. However, some analysis of the security of the cipher has led to the conclusion that it is far from secure due to its relatively short LFSRs and the fact that the feedback connections are fixed. A simple attack was proposed by Dr. Ross Anderson [34] that reduces the effective key length to just 41 bits by guessing the contents of $LFSR_1$ and $LFSR_2$, determining the contents of $LFSR_3$ from examination of the keystream and then continuing on to determine if the guesses were in fact correct. Other attacks are said to exist but they are still awaiting publication.

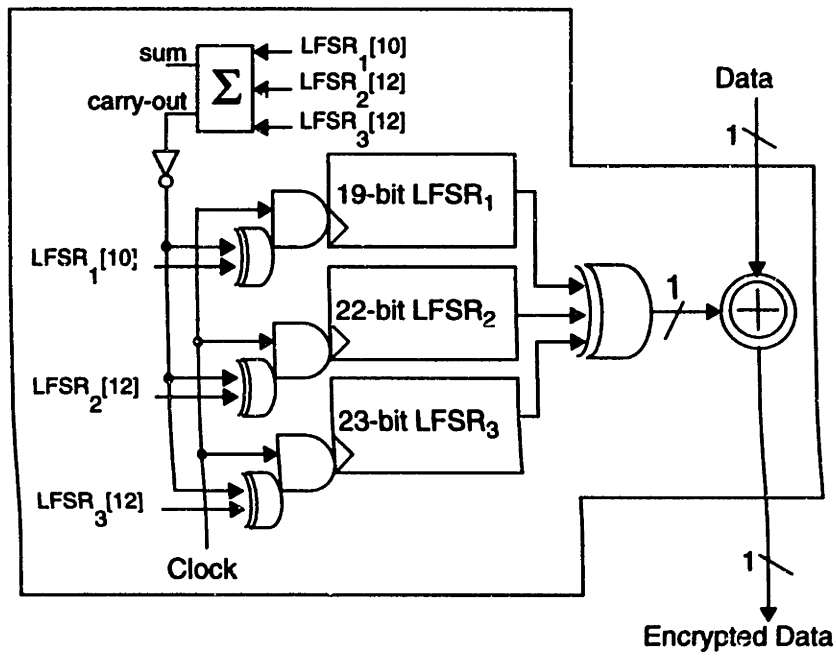



**Figure 3-5:** GSM Cipher (i.e., A5)

According to Schneier, the basic structure and underlying construction of A5 look very promising [33]. However, to be truly effective, the LFSRs must be made longer and the feedback polynomials must be made programmable. In its current form A5 provides only a marginal level of security.

## 3.5 Estimating the Power Consumption

The various LFSR-based ciphers differ in terms of their power consumption as dictated by their clock-control schemes and non-linear combining functions. The basic hardware circuit in all of the aforementioned ciphers is the LFSR, whose power consumption is represented as a function of their length (i.e., P($n$)) and the probability that they will be clocked on any given cycle (i.e., $\alpha$). A first order approximation of the power consumption of each scheme can be estimated and compared using these building blocks. Note that the ciphers must be normalized in terms of their security to provide a meaningful comparison.

The Alternating Stop and Go (ASG) Generator utilizes $LFSR_3$ to control the clocking of $LFSR_1$ and $LFSR_2$, whose outputs are then XORed to produce the keystream. On any given cycle, the probability of clocking $LFSR_1$ is 1/2 and the probability of clocking $LFSR_2$ is 1/2 as well. However, only one of the two can ever be clocked on any given

cycle, hence $\alpha_1 = \alpha_2 = 1/2$, and $\alpha_3 = 1$. The combining function is the XOR of the outputs of $LFSR_1$ and $LFSR_2$, both of which are assumed to be equally likely a 1 or 0, therefore $\alpha_{out} = 1/2$. Combining these results, a preliminary power estimate for the ASG is given by the expression:

$$\begin{aligned} P_{ASG} &= \alpha_1 \cdot P(l_1) + \alpha_2 \cdot P(l_2) + \alpha_3 \cdot P(l_3) + \alpha_{out} \cdot P(xor) \\ &= \frac{1}{2}(P(l_1) + P(l_2)) + P(l_3) + \frac{1}{2}P(xor) \end{aligned}$$

where $l_1$, $l_2$, and $l_3$ are the lengths of $LFSR_1$, $LFSR_2$, and $LFSR_3$ respectively.

The Shrinking Generator utilizes $LFSR_{sel}$ to determine whether or not the output of $LFSR_{gen}$ is to be used in the output keystream. Since there is no guarantee that a valid output will occur on any given cycle, the clock rate of the generator must be increased and an output buffer utilized to ensure that the probability of not having a valid output bit being output on any given cycle is sufficiently low. This increased clock rate is indicated by the multiplication factor $k$. Note that the power consumption of the output buffers are not accounted for in this analysis as all of the generators will implicitly require output buffering in order to decouple them from the data stream. The estimated power consumption of the Shrinking Generator is given by the expression:

$$\begin{aligned} P_{SG} &= k \cdot \alpha_{sel} \cdot P(l_{sel}) + k \cdot \alpha_{gen} \cdot P(l_{gen}) \\ &= k(P(l_{sel}) + P(l_{gen})) \end{aligned}$$

where $l_{sel}$ and $l_{gen}$ are the lengths of $LFSR_{sel}$ and $LFSR_{gen}$ respectively.

The Self-Shrinking Generator requires only a single LFSR whose clock rate must be double that of the Shrinking Generator. The power consumption of the output buffers is once again ignored. The estimated power consumption of the Self-Shrinking Generator is:

$$\begin{aligned} P_{SSG} &= 2 \cdot k \cdot \alpha \cdot P(l) \\ &= 2 \cdot k \cdot P(l) \end{aligned}$$

where $l$ is the length of the LFSR.

The GSM Cipher uses 3 LFSRs ($LFSR_{19}$, $LFSR_{22}$, and $LFSR_{23}$) whose clock control is derived from the summation of the middle bits of the LFSRs. The clocking probabilities of all 3 LFSRs is $\alpha = 3/4$, as derived using Table 3-1 and the output is the XOR of the three LFSRs' outputs. Using these results the power consumption of the GSM Cipher can be estimated to be:

$$P_{SG} = \alpha_{19} \cdot P(19) + \alpha_{22} \cdot P(22) + \alpha_{22} \cdot P(23) + \alpha_{out} \cdot P(xor3)$$
$$= \frac{3}{4}(P(19) + P(22) + P(23)) + \frac{1}{2}P(xor3)$$

| Middle Bits | | | Clock Enable | | |
|---|---|---|---|---|---|
| $LFSR_{19}$ | $LFSR_{22}$ | $LFSR_{23}$ | $LFSR_{19}$ | $LFSR_{22}$ | $LFSR_{23}$ |
| 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 1 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 |

**Table 3-1:** Clocking function for the GSM Cipher

Experimentation using EPIC's Powermill power estimation tool has shown (Figure 3-6) that to a first order approximation, the power consumption for an LFSR is a linear function that scales directly with its length (i.e., $z(P(x) + P(y)) = P(z(x + y))$). Hence, the aforementioned power estimates can be directly compared to determine which is the most efficient algorithm. For the comparison, the effective key length was chosen to be 40 bits as it is the effective key length of A5. Since the effective key length of A5 is fixed and cannot be changed, all other algorithms were scaled accordingly.

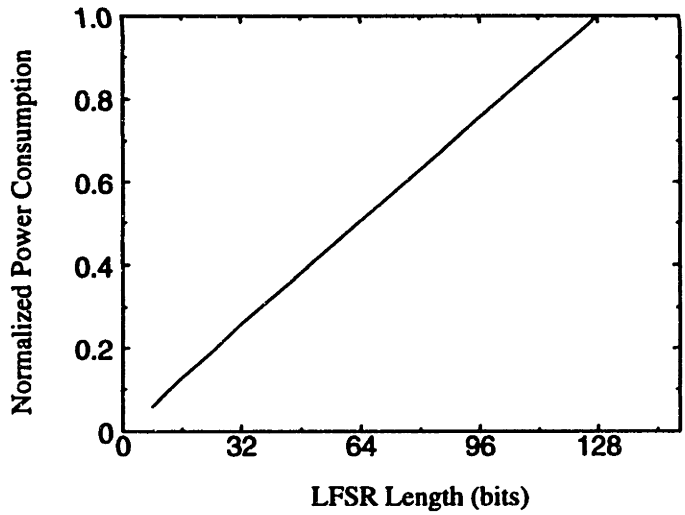



**Figure 3-6:** Normalized Power Consumption as a Function of LFSR Length

Table 3-2 summarizes the results of this comparison and shows that A5 provides the most efficient implementation in terms of power consumption. Note that the power consumption of any output combining function XORs is minimal in comparison to the power consumption of the shift registers and hence it has been ignored.

| Algorithm | Parameters | Estimated Power |
|---|---|---|
| Alternating Stop & Go | $l_1 = 31, l_2 = 32, l_3 = 41$ | P(72) |
| Shrinking Generator | $l_{gen} = 37, l_{sel} = 41, k = 2.5$ | P(195) |
| Self-Shrinking Generator | $l = 50, k = 2.5$ | P(250) |
| A5 | | P(48) |

**Table 3-2:** Power Consumption Estimates

Simulations on the fully extracted layouts of the 4 stream ciphers were performed using Powermill. They show that the above estimates are quite accurate (Figure 3-7), differing by at most 16%.

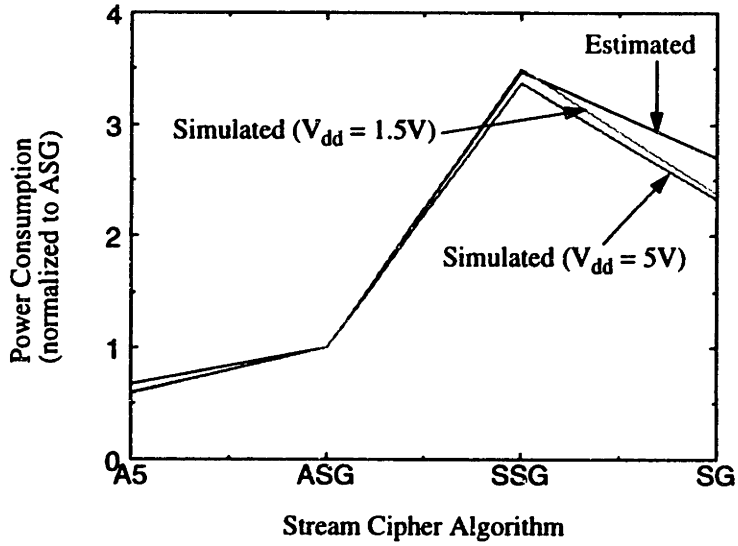



**Figure 3-7:** Comparison of Power Consumption (Estimate vs. Powermill Simulations)

## 3.6 Reducing the Power Consumption

### 3.6.1 Conventional Supply Scaling

Analysis of the various LFSR-based stream ciphers leads to the conclusion that their critical path is short enough that they can satisfy the 1 Mbps data throughput requirement while operating at the minimum allowable supply voltage of 1.5V.

### 3.6.2 Parallelizing the LFSRs

With the supply voltage minimized, the power consumption can be reduced further still by reducing the clock frequency of the LFSRs by parallelizing the shift registers used to construct them. An example of a 2-way parallel shift register implementation is shown in Figure 3-8. The parallel shift register utilizes a half rate clock but requires additional multiplexing and routing. Higher degrees of parallelism will also require control circuitry to generate multiple clock phases for each of the parallel registers. The multiplexor, routing and clock generation circuitry overhead is ultimately what determines the optimal degree of parallelization (Figure 3-9 [35]).

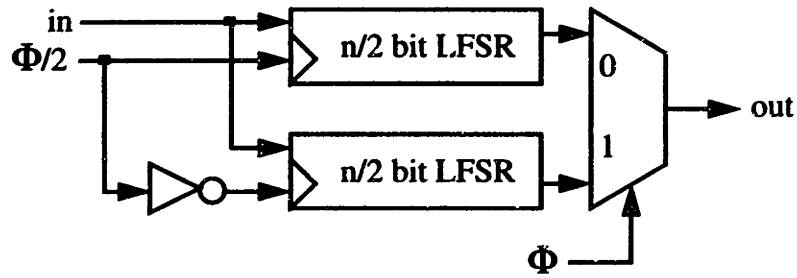



**Figure 3-8:** 2-way Parallel Shift Register

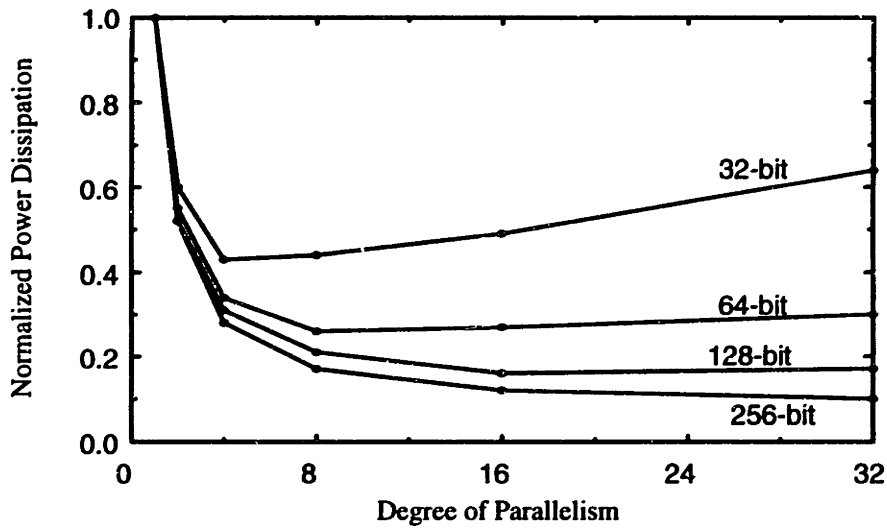



**Figure 3-9:** Degree of Parallelism vs. Normalized Power Consumption for Shift Registers

Parallelizing LFSRs is not quite so straightforward. The feedback tap positions must change their relative positions with time as the location of bit *i* within the LFSR changes from cycle to cycle. In general, for an *n*-way parallelized LFSR, the location of bit *i* can be any of *n* positions. The additional multiplexing required for the feedback taps increases the overhead dramatically (Figure 3-10), making the optimum parallelization quite different from that of the simple shift register. Analysis using 520-bit matched filters with 32 feedback taps showed that the optimal amount degree of parallelism is only 2 [35].

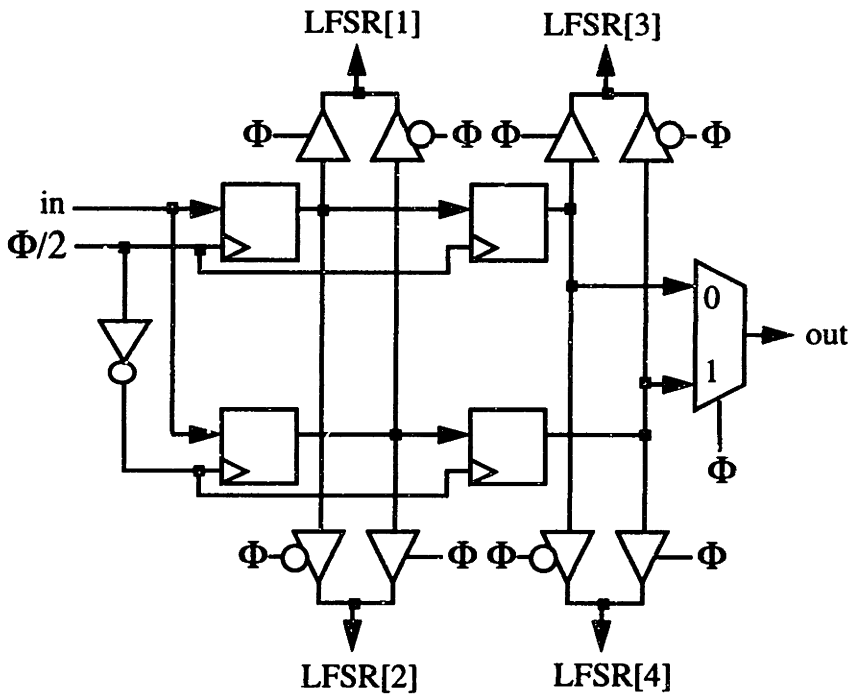



**Figure 3-10:** 2-way Parallelized LFSR

Unfortunately, it was not possible to implement parallel LFSRs during the layout of the Stream Cipher Test Chip due to time restrictions. A second-generation stream cipher test chip that will utilize the techniques described above is currently in development.

## 3.7 Stream Cipher Test Chip Design

The four aforementioned LFSR-based stream cipher designs are quite similar in design, consisting of multiple linear feedback shift registers with algorithmic-specific output-combiner and clock-control circuitry. The design details of the output combiner and clock-control circuitry are described in this section, along with the design of the variable length LFSRs with programmable feedback polynomials used in three of the four stream cipher designs.

A full set of schematics for all four LFSR-based stream ciphers can be found in Appendix A.

### Variable Length LFSRs With Programmable Feedback

The basic building block of three of the four stream cipher designs is a variable length LFSR with programmable feedback. These LFSRs were constructed using a variable

length shift register and programmable feedback tree (Figure 3-11).

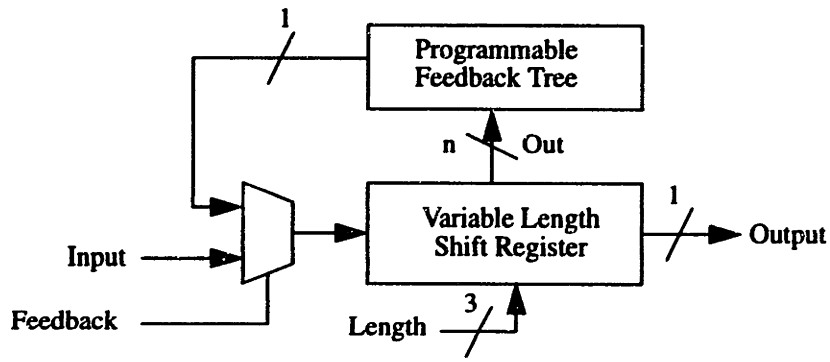



**Figure 3-11:** Variable Length LFSR Block Diagram

The variable length shift registers were constructed by partitioning an n-bit shift register into m-bit blocks, controlling the clock distribution to each of the m-bit blocks and using an output multiplexor for output selection. m was chosen to be 8 for this design in order to minimize the complexity of the clock decoder while still providing a reasonably fine granularity in terms of the shift register length.

Three separate values of n were chosen (63,64, and 65) in order to provide a large number of relatively prime register length combinations as required by the aforementioned LFSR algorithms. This variation in length is implemented using either a 7, 8 or 9 bit shift register as the first shift register in the chain. The clock decoder consists of a 3-to-8 thermometer decoder whose outputs serve as clock gating signals for each 8-bit section of the shift registers. The output of the shift register is selected using an 8-to-1 multiplexor controlled by the same 3 control bits that are used by the clock decoder (Figure 3-12).

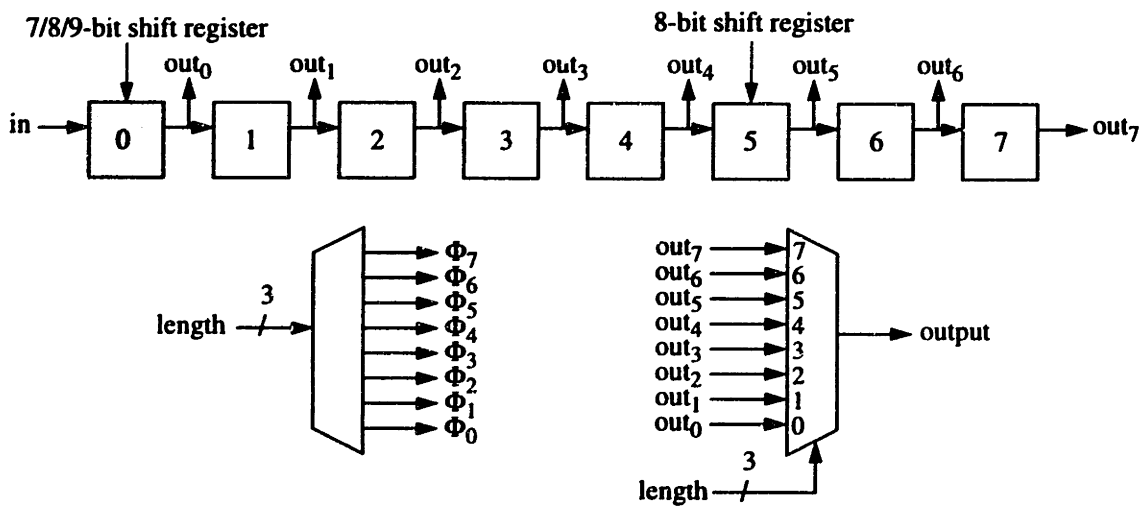



**Figure 3-12:** Variable Length Shift Register Block Diagram

The programmable feedback tree consists of a polynomial register and XOR tree. Each bit of the polynomial register is used to mask a bit in the variable length shift register such that each input to the XOR tree is a bitwise-ANDing of the shift register bit and its corresponding bit in the feedback polynomial register. The results of this ANDing are then summed modulo-2 using a $\log_2 n$ depth XOR tree (Figure 3-13) and the resultant bit is fed back to the shift register as its input when operating in LFSR mode (Figure 3-11).

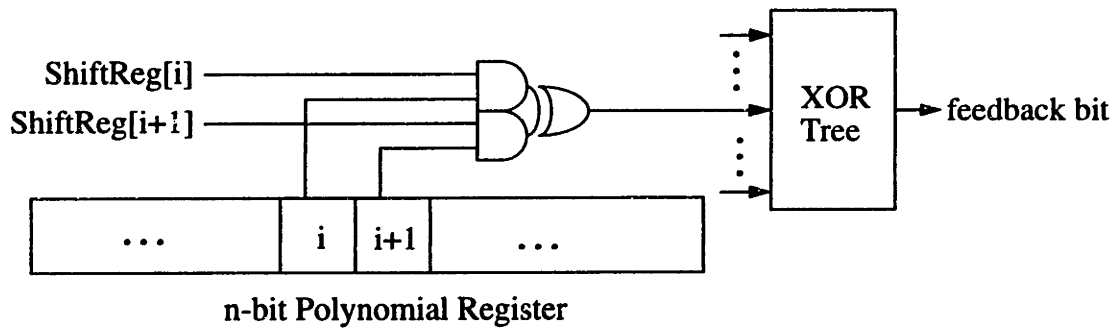



**Figure 3-13:** Programmable Feedback Tree Block Diagram

### Shrinking Generator

The SG (Figure 3-1) was implemented using a 64 bit variable length LFSR for $LFSR_{gen}$ and a 65-bit variable length $LFSR_{sel}$. The output consists of both a data output and a dataValid output. The dataValid output is the output of the $LFSR_{sel}$ and is used to enable the output latching of $LFSR_{gen}$'s output bit (Figure 3-14). dataValid is registered on the falling edge of $\Phi$ to ensure that the clock mask signal overlaps the high phase of $\Phi$. This ensures that no glitching occurs on the gated clock (Figure 3-15).

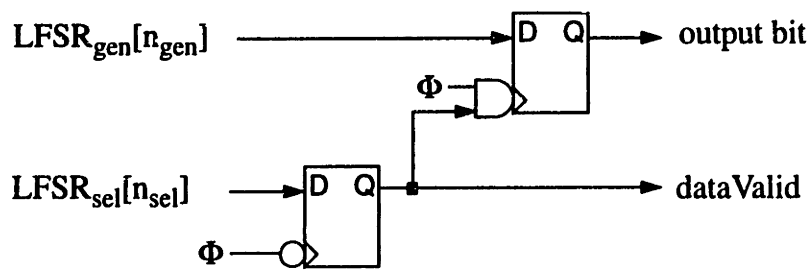



**Figure 3-14:** Shrinking Generator Output Latching

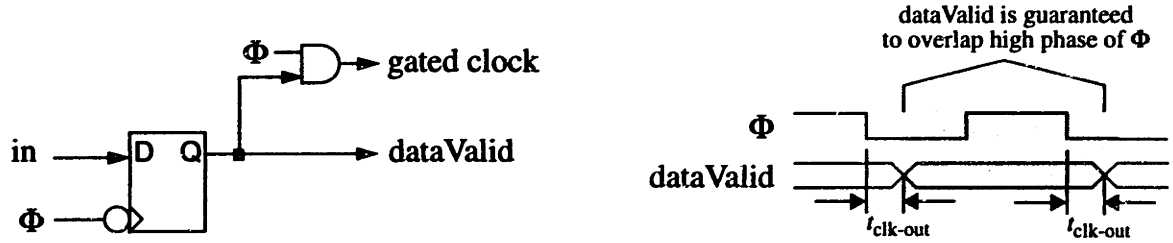
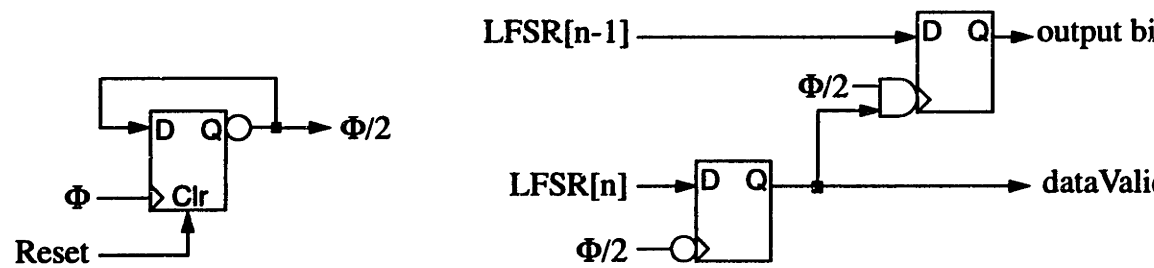



**Figure 3-15:** Clock Gating

### Self-Shrinking Generator

The SSG (Figure 3-3) utilizes a single 65-bit variable length LFSR whose two most significant bits are fed to an output selection circuit that operates at half of the clock rate of the LFSR. The output selection circuit (Figure 3-16) generates a half frequency clock using a resetable toggle flip-flop and then uses this clock to drive the output latch circuitry. Once again, dataValid is registered on the falling edge of Φ to ensure that no glitching occurs on the gated clock.




(a) clock divider (b) output latch circuit

**Figure 3-16:** Self-Shrinking Generator Output Circuit

### Alternating Stop and Go Generator

The ASG (Figure 3-4) was implemented using three variable length shift registers of lengths 63, 64, and 65 bits for $LFSR_1$, $LFSR_2$, and $LFSR_3$ respectively. Clock control is done using an active-high enable clock buffer for $LFSR_1$ and an active-low enable clock buffer for $LFSR_2$, where the enable signal is the output of $LFSR_3$. The output is taken from the XORing of the outputs of $LFSR_1$ and $LFSR_2$.

### A5 Cipher

The A5 cipher (Figure 3-5) was implemented using three fixed-length LFSRs of length 19, 22, and 23 bits with variable feedback connections. The clock control function was implemented using three XNOR gates and three OR gates to generate the three clock enable signals (Figure 3-17). The output is taken from the XORing of the three LFSRs' output bits.

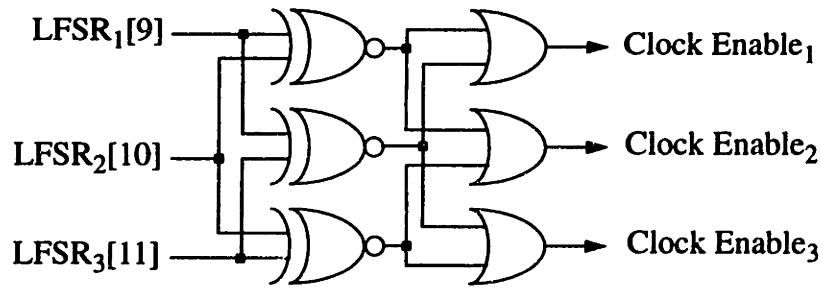



**Figure 3-17:** A5 Clock Control Circuit

### Layout

The final stream cipher test chip design is shown in Figure 3-18. The entire test chip contained a total of 27,910 devices and consumed a total area of 18.38mm$^2$ (4.29mm x 4.29mm). The design is severely pad-limited, the core logic that makes up the 4 stream ciphers consumes only 19% of the total chip area due to the large number of pads required to test each stream cipher as an individual unit. The physical statistics of the individual ciphers are given in Table 3-3

| Design | Device Count | Area |
|---|---|---|
| Shrinking Generator | 7,998 | 978,144 $\mu m^2$ |
| Self-Shrinking Generator | 4,167 | 518,910 $\mu m^2$ |
| Alternating Stop & Go Generator | 11,923 | 1,477,039 $\mu m^2$ |
| GSM Cipher (A5) | 3,822 | 495,040 $\mu m^2$ |

**Table 3-3:** Layout Statistics for Stream Cipher Test Chip

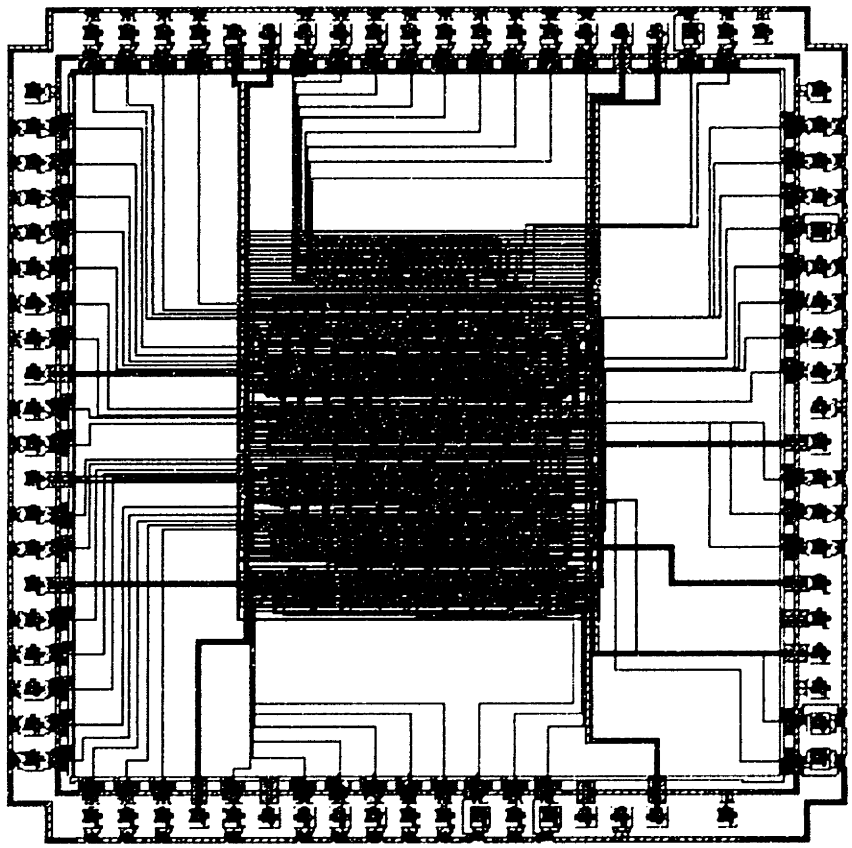


**Figure 3-18:** Stream Cipher Test Chip Layout

## 3.8 Preliminary Test Results

Initial testing has verified that the Stream Cipher Test Chip is fully functional. The testing was performed over a range of supply voltages from 5V down to a minimum of 1.1V[3] at a data rate of 1 Mbps. For supply voltages below 1.1V, all of the circuits ceased to function.

Preliminary power consumption estimates were obtained by using EPIC's Powermill power estimation tool with the extracted physical layout of the test chip (Table 3-4). Actual power consumption measurements were then performed with the Stream Cipher Test Chip (Table 3-4 and Figure 3-19) and they match quite favourably with the Powermill results at a supply voltage of 5V. At low supply voltages (i.e., 1.5V) the measured and estimated power begin to deviate, due, most likely, to the inability of Powermill and the underlying hspice models to accurately simulate subthreshold device operation.

---

3. only the ASG and A5 operated @1 Mbps at $V_{dd}$ = 1.1V, the SSG operated @1 Mbps down to 1.2V, and the SG operated @1 Mbps down to 1.15V

| Algorithm | Estimated Power Consumption (μW) | | Measured Power Consumption (μW) | |
|---|---|---|---|---|
| | $V_{dd}$ = 5V | $V_{dd}$ = 1.5V | $V_{dd}$ = 5V | $V_{dd}$ = 1.5V |
| Alternating Stop & Go Generator | 395.6 | 27.4 | 395 | 33 |
| Shrinking Generator | 920.3 | 65.0 | 925 | 65 |
| Self-Shrinking Generator | 1333.2 | 85.0 | 1335 | 108 |
| A5 | 232.0 | 16.4 | 215 | 17 |

**Table 3-4:** Estimated and Measured Power Consumption of the 4 LFSR-based Stream Ciphers at 1 Mbps

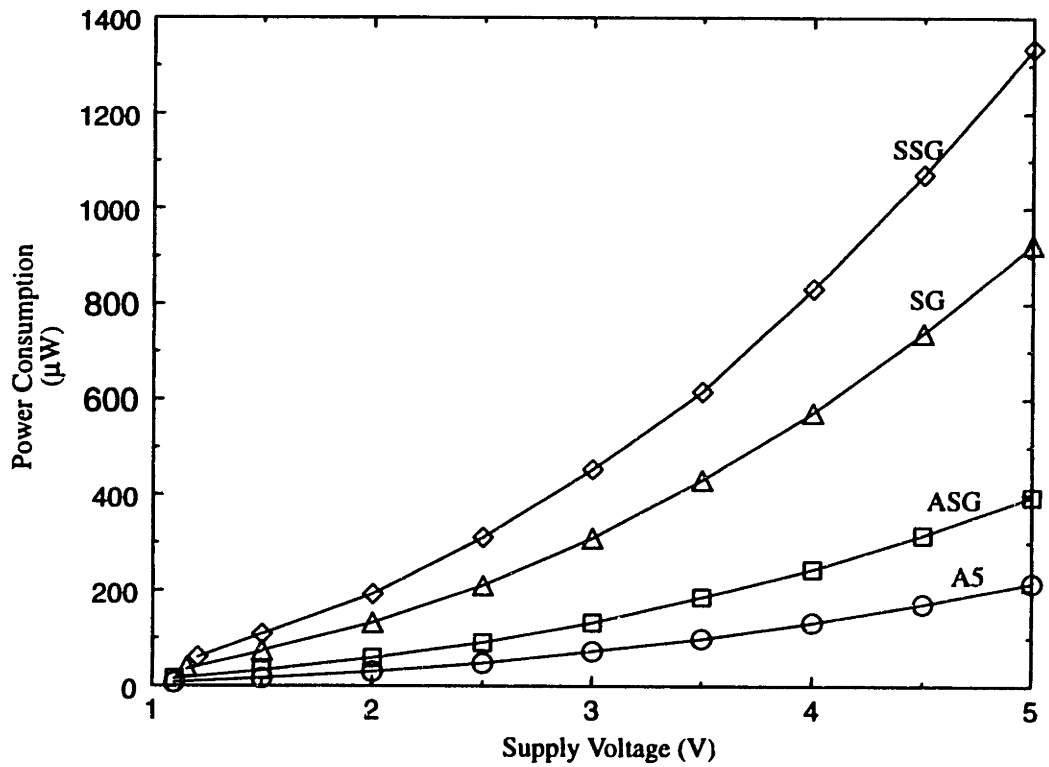



**Figure 3-19:** Measured Power Consumption of the 4 LFSR-based Stream Ciphers

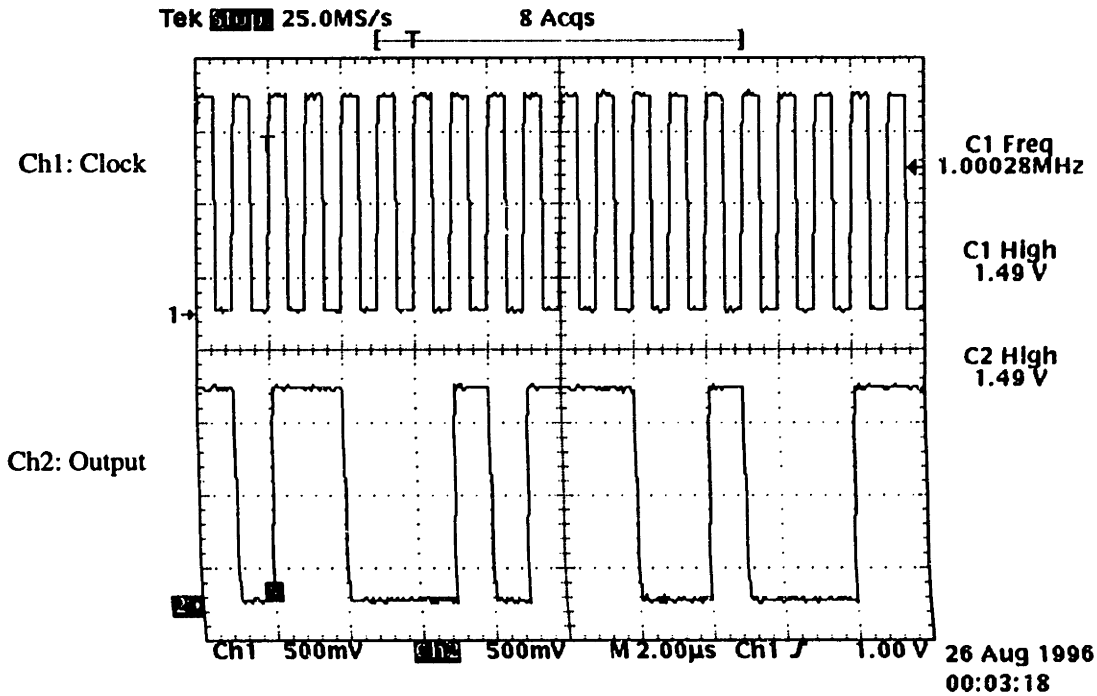



**Figure 3-20:** A5 Stream Cipher Operation @$V_{dd}$ = 1.5V

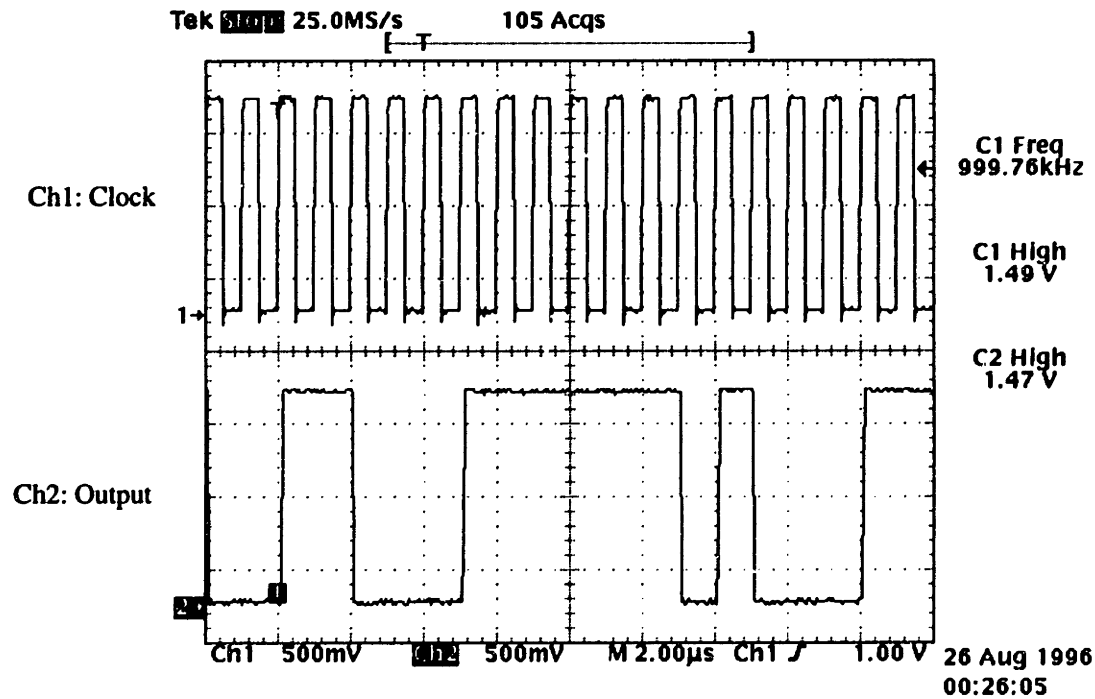



**Figure 3-21:** ASG Stream Cipher Operation @$V_{dd}$ = 1.5V

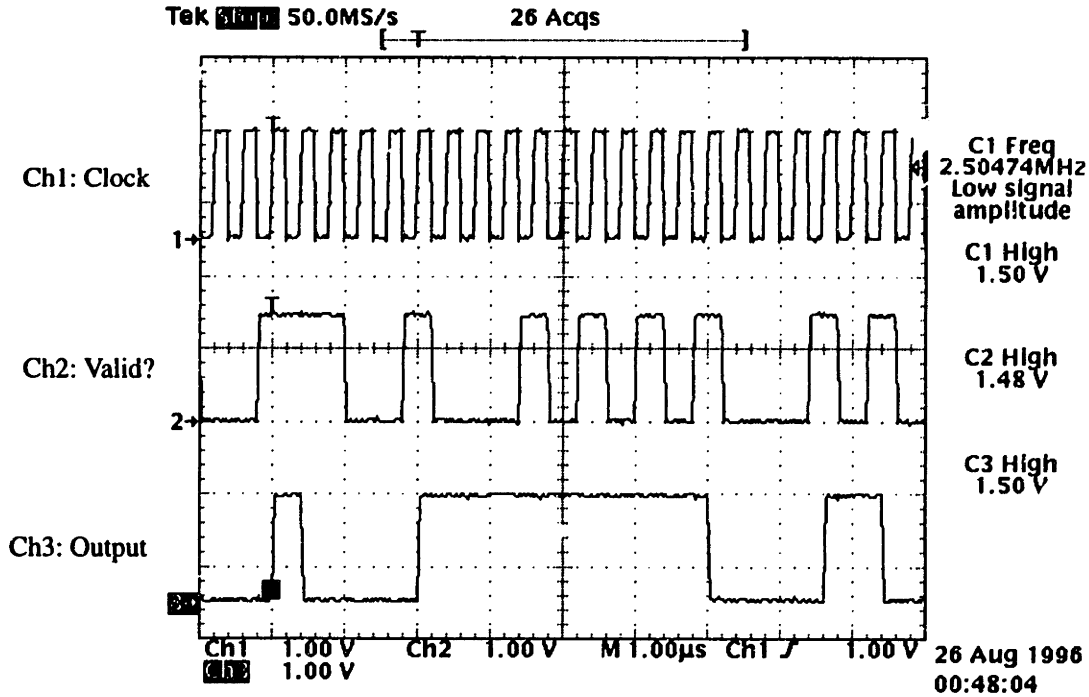



**Figure 3-22:** SG Stream Cipher Operation @$V_{dd}$ = 1.5V

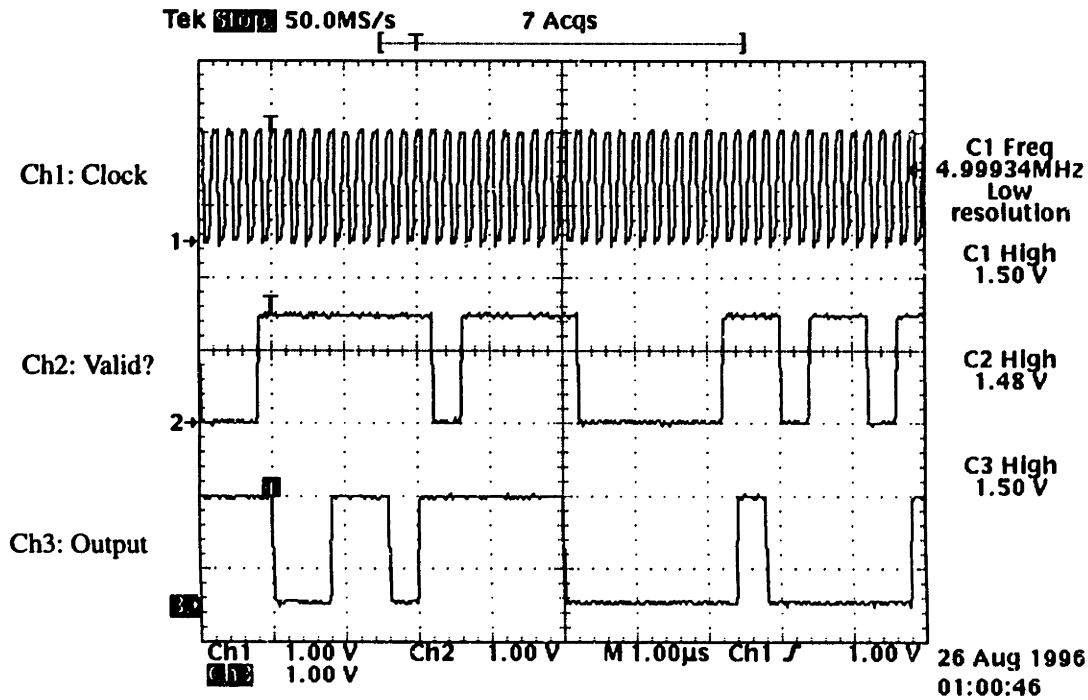



**Figure 3-23:** SSG Stream Cipher Operation @$V_{dd}$ = 1.5V

# Chapter 4

# QRG-Based Stream Cipher Systems

The QRG-based stream cipher utilizes a QRG to generate a provably secure pseudo-random sequence that is used as the keystream which is then XORed with the data stream to form the encrypted data stream. The fact that the QRG-based stream cipher is provable secure makes it a very attractive algorithm. Unfortunately, it is quite complex and can be too computationally intensive for an ultra low power application. Care must be taken in selecting an efficient algorithm for the implementation of the modular squaring operation at the heart of the QRG in order to minimize the power consumption.

In this section some basic properties of the QRG are first discussed and then various algorithms for implementing the QRG are compared to determine their suitability to a low-power implementation. Various power reduction techniques are then applied to reduce the power consumption even further. Finally, a hardware implementation that was fabricated is described and test results are presented.

## 4.1 Some Basic Number Theory

Understanding the subsequent discussions on the QRG requires knowing some basic results from number theory. These results are presented here without proof as reference. Note that it is assumed that the reader is familiar with basic concepts such as finite fields and the basic properties of modular arithmetic.

**Group Order**

Given a finite field $\mathbf{Z}_p = \{0,1,...,p\text{-}1\}$, the order of an element ($\alpha$) of the field is defined as the smallest non-negative integer $t \geq 1$, such that $\alpha^t = 1 \bmod p$, and is commonly expressed as $\mathrm{ORD}_p\alpha = t$.

**Multiplicative Groups Modulo *n***

A multiplicative group modulo $n$, $\mathbf{Z}_n^*$, is the subgroup of $\mathbf{Z}_n$ containing all $a \in \mathbf{Z}_n$ such

that gcd($a$,$n$) = 1. The number of elements in $\mathbf{Z}_n^*$ is given by Euler's totient function:

$$\phi(n) = n\prod_{p|n}\left(1-\frac{1}{p}\right)$$

**Quadratic Residues**

Given an odd prime integer $p$ and an integer $0 < x < p$, $x$ is a quadratic residue modulo $p$ if there exists some $y \in \mathbf{Z}_p$ such that $y^2 = x \bmod p$. The set of quadratic residues modulo $p$ is often denoted as $\mathbf{QR}_p$. Euler showed that $x$ is a quadratic residue if and only if:

$$x^{(p-1)/2} \equiv 1 \bmod p$$

This is known as Euler's Criterion.

**Legendre Symbols**

Given an odd prime integer $p$ and an integer $a \geq 0$, the Legendre symbol $\left(\frac{a}{p}\right)$ is defined as:

$$\left(\frac{a}{p}\right) = \begin{cases} 0, \text{ if a} = 0 \bmod \text{p.} \\ 1, \text{ if a is a quadratic residue modulo p.} \\ -1, \text{ if a is a quadratic non-residue modulo p.} \end{cases}$$

from which it follows that:

$$\left(\frac{a}{p}\right) \equiv a^{(p-1)/2} \bmod p$$

**Jacobi Symbols**

Jacobi symbols are a generalization of Legendre symbols for the case where $p$ is positive odd integer whose factorization is $p = p_1^{e_1} \cdot p_2^{e_2} \cdot \ldots \cdot p_k^{e_k}$. Given $p$ and an integer $a \geq 0$, the Jacobi symbol $\left(\frac{a}{p}\right)$ is defined to be:

$$\left(\frac{a}{p}\right) = \prod_{i=1}^{k}\left(\frac{a}{p_i}\right)^{e_i}$$

Despite the above definition, techniques do exist to compute $\left(\frac{a}{p}\right)$ without knowing the factorization of $p$ (e.g. [36]).

**Carmichael Numbers**

Given an integer $N \geq 1$ whose factorization is $N = 2^e \cdot P_1^{e_1} \cdot \ldots \cdot P_k^{e_k}$, the Carmichael number $\lambda(N)$ is defined to be:

$$\lambda(N) = lcm\left(\lambda\left(2^e\right), (P_1 - 1) \cdot P_1^{e_1 - 1}, \ldots, (P_k - 1) \cdot P_k^{e_k - 1}\right)$$

where:

$$\lambda\left(2^e\right) = \begin{cases} 2^{e-1}, \text{if e} = 1 \text{ or } 2. \\ 2^{e-2}, \text{if e} > 2. \end{cases}$$

**Carmichael's Extension to Euler's Theorem**

Euler originally proved that for any integer $n \geq 1$, and all $a \in \mathbf{Z}_n^*$:

$$a^{\phi(n)} \equiv 1 \bmod n$$

Carmichael extended this result by proving that given any $a \in \mathbf{Z}_n^*$ such that $\gcd(a,n) = 1$, then:

$$a^{\lambda(n)} \equiv 1 \bmod n$$

## 4.2 Security of the QRG

Unlike the aforementioned LFSR-based Stream Ciphers the QRG is a provably secure generator in the sense that even given all previous output bits, an attacker has no better than a 50% chance of predicting the next output bit assuming that they don't know how to factor the modulus N.

In their original work, Blum, Blum, and Shub stated that the BBS Generator, upon which the QRG is based, was cryptographically secure modulo the Quadratic Residue Assumption which states that any efficient procedure that guesses whether or not a number is a quadratic residue will be incorrect some fraction of the time [17]. Vazirani and Vazirani extended the work to show that breaking the BBS Generator is as difficult as factoring its modulus N [37]. Given the factorization of N, an attacker could quickly and efficiently compute the Legendre Symbols of any given $x_i$ (i.e., $\left(\frac{x_i}{p}\right)$ and $\left(\frac{x_i}{q}\right)$, where N = p·q) to determine if $x_i$ were in fact a Quadratic Residue.

For moduli whose length are on the order of 512 bits or more this is believed to be a computationally difficult problem (in Section 2.6 we quantified just how difficult this problem is to solve).

In [36], Stinson takes another approach and proves that cracking the BBS Generator is computationally equivalent to an unbiased, polynomial-time Monte Carlo algorithm for Quadratic Residues having error probability at most $\delta$, for any $\delta > 0$, a problem that is believed to be intractable.

## 4.3 Period of the QRG

Calculating the period of the QRG requires that both the modulus, $N$, and the initial value, $x_0$, be chosen in a specific way. The period of the QRG can be calculated using the following two properties [17], which will be subsequently derived:

1. if $N = P \cdot Q$, where $P \equiv Q \equiv 3 \bmod 4$, and $x_0$ a quadratic residue modulo $N$ then $\pi(x_0) \mid \lambda(\lambda(N))$, where $\lambda(y)$ is the Carmichael number of $y$.
2. if $x_0$ is chosen such that $\mathrm{ORD}_N(x_0) = \lambda(N)/2$, then $\lambda(\lambda(N)) \mid \pi(x_0)$.

The first property can be derived by first applying Carmichael's extension of Euler's theorem to determine that $2^{\lambda(\mathrm{ORD}_N x_0)} = 1 \bmod (\mathrm{ORD}_N x_0)$. Since $\pi(x_0)$ is the period of the QRG it follows that $x_0 = x_0^{2^\pi} \bmod N$, and thus $x_0^{2^\pi - 1} = 1 \bmod N$. Hence $2^\pi - 1 = \mathrm{ORD}_N x_0$ which implies that $2^\pi = 1 \bmod (\mathrm{ORD}_N x_0)$ and it follows that:

$$\pi \mid \lambda(\mathrm{ORD}_N x_0) \tag{4-1}$$

Using Carmichael's extension again we see that $x_0^{\lambda(N)} = 1 \bmod N = x_0^{(\mathrm{ORD}_N x_0)}$ and thus:

$$\mathrm{ORD}_N x_0 \mid \lambda(N) \tag{4-2}$$

Combining EQ 4-1 and EQ 4-2 yields the expression:

$$\pi(x_0) \mid \lambda(\lambda(N)) \tag{4-3}$$

If $x_0$ is chosen such that $\mathrm{ORD}_N(x_0) = \lambda(N)/2$ then $x_0^{\lambda(N)/2} = 1 \bmod N$. As previously stated, $x_0^{2^\pi - 1} = 1 \bmod N$, hence $2^\pi - 1 = \lambda(N)/2$ and:

$$2^{\pi} = 1 \bmod (\lambda(N)/2) \tag{4-4}$$

If $N$ is chosen such that $N = PQ = (2p_1+1)(2q_1+1) = (2(2p_2 + 1) + 1)(2(2q_2+1)+1) = (4p_2 + 3)(4q_2+3)$ (which guarantees that P = Q = 3 mod 4 as required) where $P$, $Q$, $p_1$, $q_1$, $p_2$ and $q_2$ are all odd primes then it can be shown that $\mathrm{ORD}_{\lambda(N)/2}(2) = \lambda(\lambda(N))$ [17]. Thus $\lambda(\lambda(N))$ is the least positive integer such that:

$$2^{\lambda(\lambda(N))} = 1 \bmod (\lambda(N)/2) \tag{4-5}$$

Combining EQ 4-4 and EQ 4-5 yields the expression:

$$\lambda(\lambda(N)) \mid \pi(x_0) \tag{4-6}$$

Combining EQ 4-3 and EQ 4-6 leads to the desired expression for the period of the QRG:

$$\pi(x_0) = \lambda(\lambda(N)) \tag{4-7}$$

Given an n-bit modulus of the prescribed form, a rough estimate of the period of the QRG can be calculated using EQ 4-7. Recall that $N = PQ$, where $P = 2p_1 + 1$, $Q = 2q_1 + 1$, $p_1 = 2p_2 + 1$ and $q_1 = 2q_2 + 1$. Thus the period of the QRG is:

$$\pi(x_0) = \lambda(\lambda(N))$$

$$\lambda(N) = \mathrm{lcm}\{(P-1), (Q-1)\} = \mathrm{lcm}\{2p_1, 2q_1\} = 2p_1q_1 = M$$

$$\lambda(M) = \mathrm{lcm}\{\lambda(2), p_1, q_1\} = \mathrm{lcm}\{1, 2p_2, 2q_2\} = 2p_2q_2$$

$$\therefore \pi(x_0) = 2p_2q_2 \tag{4-8}$$

For an n-bit modulus, $N$, with factors $P$ and $Q$ that are roughly the same size (i.e., $n/2$ bits) the size of $p_2$ and $q_2$ will be roughly ($n/2$ - 2) bits. For $n$ = 512 bits, the corresponding period is on the order of $2^{509}$ which will be more than sufficient for any practical implementation.

In terms of practical tests, the parameters of the QRG can be chosen using the following algorithms:

Generating $N$, $P$, $Q$, $p_1$, $q_1$, $p_2$, and $q_2$:

- Select odd integers $p_2$, $q_2$ of length ($n/2$ - 2) and test for primality. If both pass then proceed, otherwise continue choosing values of $p_2$ and $q_2$ until both pass
- Generate $p_1$ and $q_1$ from $p_2$ and $q_2$ and test for primality. If both pass then proceed. If any fail then repeat the previous step for that value until it satisfies both the previous and the current tests.
- Generate $P$ and $Q$ from $p_1$ and $q_1$ and test for primality. If both pass then save the values as a valid key pair. If either fails go back to the first step and repeat the entire process again until a valid value is generated.

Generating $x_0$:

- Select a random value for $x_0$ and then test that $\mathrm{ORD}_N x_0 = \lambda(N)/2$ by checking if $x_0^{p_1 q_1} = 1 \bmod N$.

## 4.4 Efficiency of the QRG

In the original definition of the QRG, it was shown that the least significant bit of the output was unpredictable [17]. However, it has since been shown that the $m = log_2(log_2(x_i))$ least significant bits of the generator's output are in fact unpredictable [37]. This improves the efficiency of the QRG by a factor $m$. The practical implications of this result are that for a 512 bit modulus, a maximum of 9 bits and an average of 8 bits can be extracted per iteration.

By way of comparison, a simple RSA encryption system, that utilizes a repeated squaring and multiplication implementation for performing modular exponentiation, requires on average $3n/2$ modular multiplications plus $n$ modular additions to generate $n$ bits of ciphertext. Thus each bit of ciphertext requires 3/2 modular multiplications and one modular addition, whereas each bit of ciphertext generated using the QRG requires at most one modular multiplication and typically $1/log_2(log_2(n))$. Hence the QRG is much more computationally efficient than the RSA encryption scheme.

## 4.5 Modular Multiplication Algorithms

The performance of the QRG depends almost entirely on the ability to quickly, and efficiently compute the square of an integer modulo some number. High speed multiplication is for the most part a simple task -- there are many very efficient algorithms available to a hardware designer (e.g., [38], [39], and [40]). However, modular multiplication is considerably more difficult and as a result, care must be taken to select and/or develop an effi-

cient algorithm that best suits the design requirements.

There are two basic approaches to performing modular multiplication: multiply then reduce and multiply with repeated partial reductions.

The first method partitions the modular multiplication into an $nxn$ bit multiplication followed by a $2n$ bit division. The belief is that by separating the two operations they can be optimized independently of one another, yielding a very efficient implementation. Unfortunately, this scheme has numerous inefficiencies: the intermediate result requires a $2n$ bit register, the $nxn$ bit multiplication will require a time consuming $2n$ bit carry propagate addition and the division circuit must be capable of handling a $2n$ bit operand. Since the latency of most arithmetic algorithms depends on the width of the operands, this results in a very slow and inefficient modular multiplication technique.

The second method utilizes approximate reductions during the accumulation of the multiplication algorithm's partial products to reduce the size of the intermediate result and the overall latency of the operation. The size of the intermediate result depends on the accuracy of the modular reduction whose latency in turn depends on the number of bits used to form the approximation. Hence, there is a trade-off to be made between latency and accuracy. Fortunately, only a moderate reduction in accuracy will yield a significant gain in latency (e.g., Takagi's algorithm [41] requires only 2 additional digits be stored, and uses only 8 digits to form the approximation). Multiplying with partial reductions is currently the algorithm of choice for implementing modular multiplication. There are many existing algorithms in the literature (e.g., [41], [42], and [43]) that utilize this approach to create very efficient implementations.

Both of the above methods typically utilize redundant number representations internally to eliminate time-consuming carry-propagation chains within the algorithm. This in turn minimizes the glitching effects that occur when a carry pulse is propagated through a long addition chain. Glitching causes energy-consuming transitions that can then propagate throughout the circuit, possibly causing a significant increase in the power consumption.

This section compares several existing algorithms in terms of their efficiency and suitability to the Quadratic Residue Generator.

### 4.5.1 Chinese Remainder Theorem

The Chinese Remainder Theorem provides a means of decomposing an operation modulo $N$, where N is a composite number whose prime factorization is $N = m_1 \cdot m_2 \cdot \ldots \cdot m_k$, into operations modulo $m_i$ ($1 \leq i \leq k$). Since most arithmetic operations' latency is proportional to the size of the operands, the latency of the overall operation can be reduced by breaking it into smaller and simpler operations.

For the modular squaring operation utilized in the QRG the n-bit modular multiplication can be decomposed into two $n$ bit modular reductions, two $n/2$ bit modular multiplications, two $n/2$ bit multiplications and a $n$ bit modular addition. These operations can be accelerated further due to the fact that they can be parallelized as shown in Figure 4-1. As we will soon see, the latency of a modular multiplication is directly proportional to the length of its modulus. Therefore, the Chinese Remainder Algorithm allows a speedup factor of 2 for the modular multiplication. However, the additional multiplication, modular reduction and modular addition, in addition to the area overhead of the parallelization make the Chinese Remainder algorithm unattractive for our application.

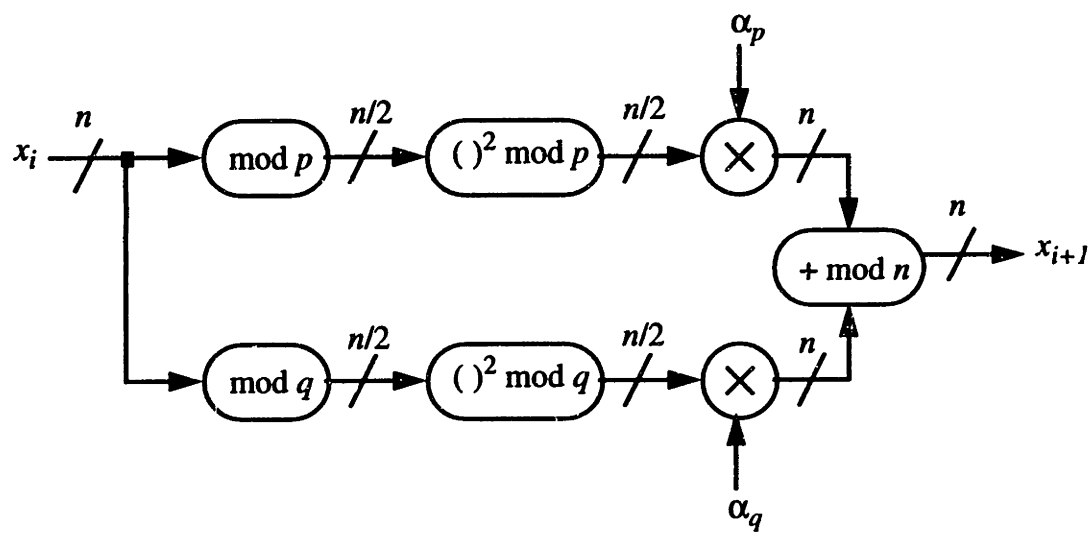



**Figure 4-1:** Parallelization of Chinese Remainder Algorithm

### 4.5.2 Morita's Algorithm

Morita proposed a high radix ($r > 2$) modular multiplication algorithm [42] that operates on two binary operands, A and B, to produce AxB modulo N, where N is an n-bit modulus. The algorithm requires approximately $\lfloor n/(\log_2 r) \rfloor + 1$ iterations to generate a

result.

The primary drawback of the Morita's Algorithm is that it requires binary inputs in non-redundant form, but it generates a result that is in a redundant form that cannot be directly fed back into the multiplier for repeated squaring. Thus the output must undergo a time-consuming carry propagate addition to convert it into a non-redundant binary representation. This significantly slows down the operating speed of the algorithm, making it unusable for our application where we require very fast, repeated modular squarings.

### 4.5.3 Orup and Kornerup's Algorithm

Orup & Kornerup's Algorithm [43] utilizes a higher radix and pipelined structure to create a fast algorithm that requires $\lceil (n+1) / (\log_2 r) \rceil + 2$ iterations, where $r$ is the radix of the multiplier, and $n$ is the length of the modulus. Each iteration requires 3 clock cycles whose length is dictated by the latency of the quotient estimation unit (QEU). The QEU computes the multiple of the modulus that is to be subtracted from the intermediate result during each iteration. Orup and Kornerup propose a fully parallelized QEU that computes all possible estimates and then selects the smallest non-negative result. For a radix-32 implementation this will require 42 separate estimates and hence 42 copies of the quotient estimation circuitry, each of which consumes valuable area and power. Any attempts to reduce the size of the QEU by reducing the number of parallel branches will result in a direct increase in the execution time of the algorithm (i.e., $T_{cycle} = T_{QEU}$ so halving the number of parallel branches doubles the cycle time). These shortcomings make Orup and Kornerup's Algorithm unattractive for our application.

### 4.5.4 Takagi's Algorithm

Takagi proposed a radix-4 modular multiplication algorithm that multiplies two redundant form numbers to generate a result that is of the same form [41]. Hence the outputs of his algorithm can be fed directly back in for repeated squarings, yielding a very efficient algorithm for implementing the QRG. In addition, the hardware requirements of Takagi's algorithm are minimal enough that it was ultimately chosen as the algorithm to be used for implementing the QRG.

A detailed description of both the algorithm and the architecture for a VLSI implementation are given below.

**The Algorithm**

Regular higher-radix multiplication algorithms typically scan their operands from LSB to MSB, forming partial products on each iteration that are then summed to generate the final result. On each iteration, $\log_2(\text{radix})$ bits of the result are generated, beginning with the LSB of the result and proceeding to the MSB. This method requires a total of $n/\log_2(\text{radix})$ iterations to form the final result of an $n$ x $n$ bit multiplication. High-radix division algorithms, on the other hand, scan the operands from MSB to LSB and generate their results beginning with the MSB and proceeding to the LSB. Takagi's algorithm utilizes the division approach of scanning from MSB to LSB to perform both an $n$-bit multiplication and modular reduction (i.e., $n$-bit division) concurrently. On each iteration, the operands are scanned from MSB to LSB to form an intermediate result whose most significant digits are then scanned to form a quotient estimate that is used to partially reduce the intermediate result. The use of a quotient estimate requires that the final result undergo one additional iteration to account for any errors introduced by the approximations in the estimation process. The algorithm requires approximately $\lfloor n/(\log_2 r) \rfloor + 1$ iterations, each of which is performed within a single clock cycle. The formal description of Takagi's algorithm is given in Figure 4-2.

**INPUTS:**

- Q: an $n$-bit binary modulus
- X: an $n$-digit redundant binary multiplicand
- Y: an $n$-digit redundant binary multiplier

**OUTPUTS:**

- P: an $n$- digit redundant binary product (i.e., $P = X \cdot Y \bmod Q$)

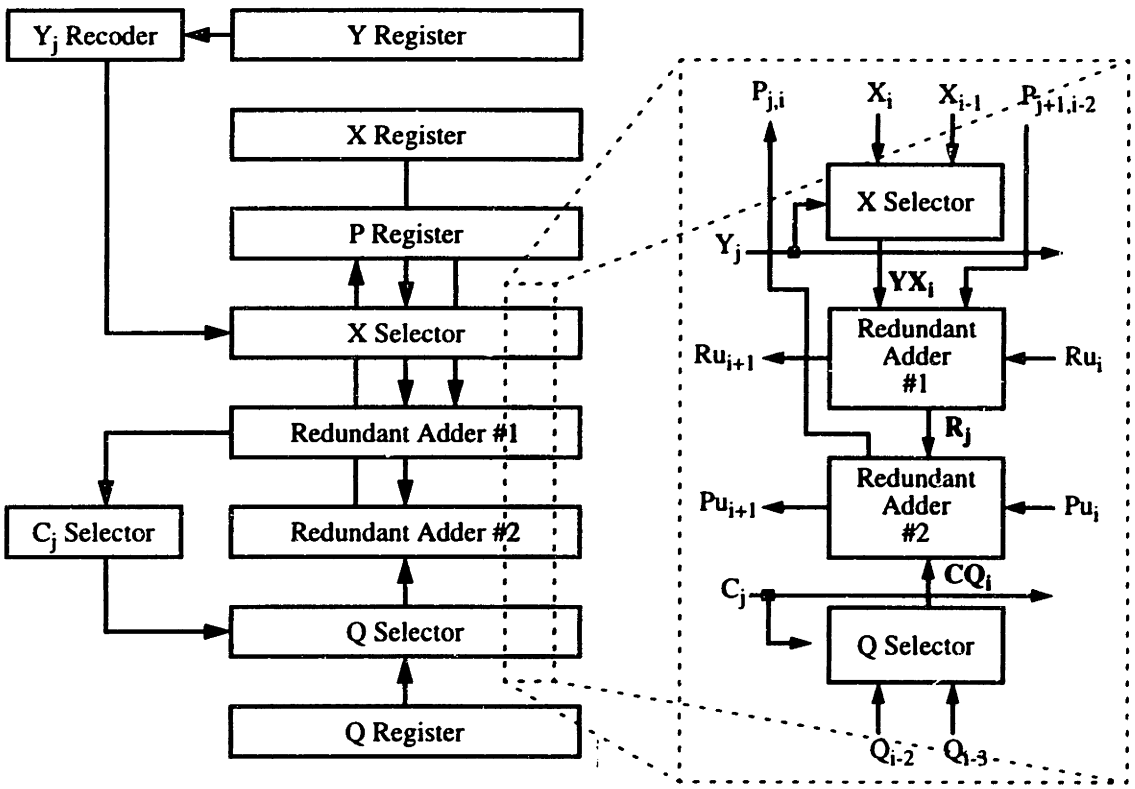
**ALGORITHM:**

1. $P_{n/2+1} = 0$
2. for j = floor($n$/2) downto -1 do
   - calculate $Y_j$ using bits 2j and (2j-1) of input Y
   - $R_j = 4 \cdot P_{j+1} + Y_j \cdot X$
   - calculate $C_j$ using the most significant digits of $R_j$ and Q
   - $P_j = R_j - 4 \cdot C_j \cdot Q$
3. $P = P_{-1} / 4$

**Figure 4-2:** Takagi's Algorithm

**The Architecture**

Takagi's algorithm is ideally suited to a VLSI implementation due to its regular cellular array structure, which can be implemented in a bitslice structure (Figure 4-3). One particularly nice feature of the architecture is that the logic depth and hence the cycle time, of the multiplier is independent of the length of the modulus due to the use of redundant number representations that eliminate long carry chains. The use of redundant number representations requires that all interconnections between the bitslices be local (i.e., very short) which eliminates the need for large global busses and their accompanying drivers. The elimination of long carry chains and global bussing allows for very short cycle times (on the order of 20-30 ns) and hence very fast modular multiplications, which is exactly what we require for implementing the QRG.



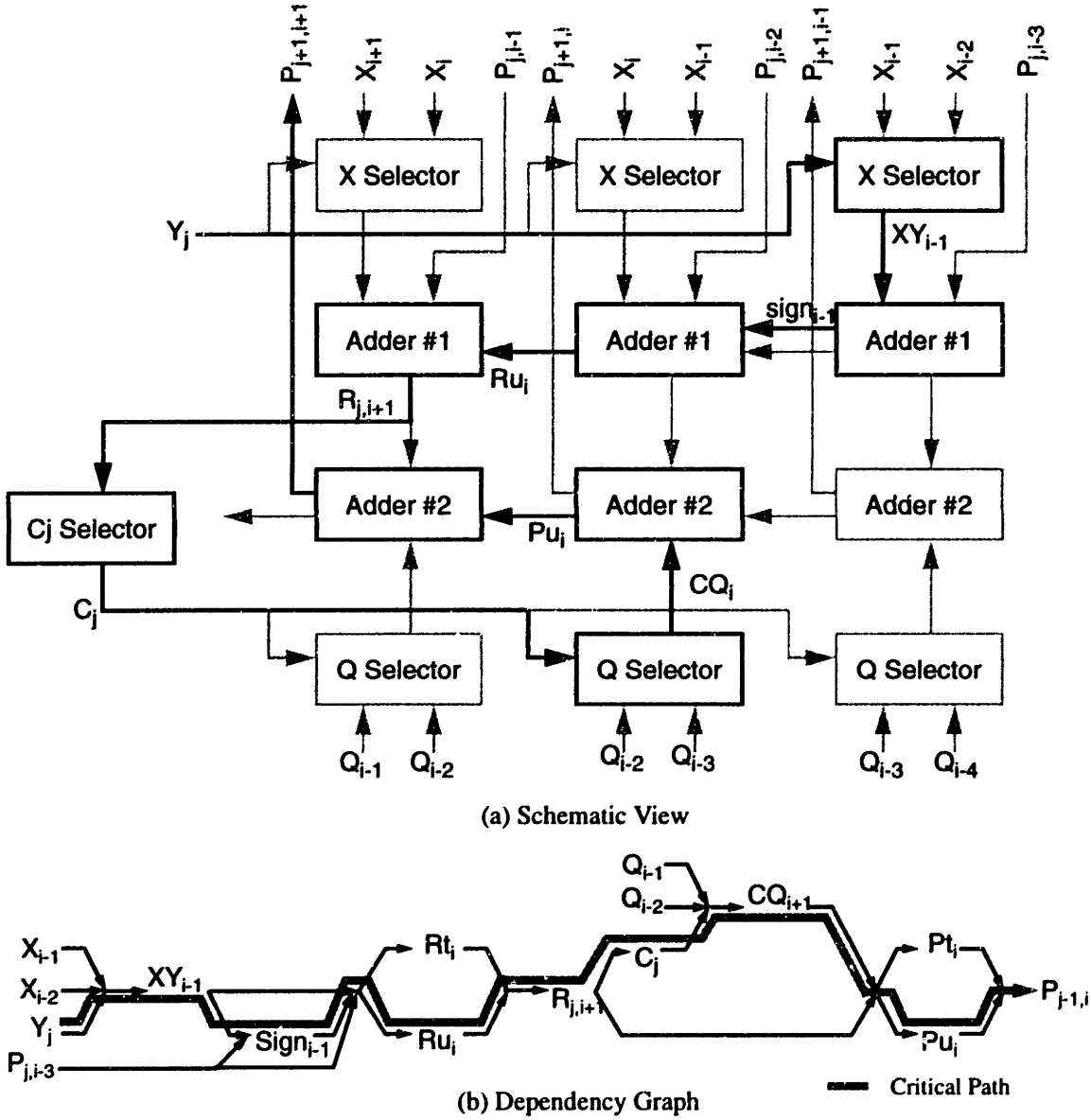

(a) Top Level Architecture (b) Bitslice i

**Figure 4-3:** Architecture of Takagi's Multiplier

## 4.6 Reducing Power Consumption

The power consumption of the QRG-based cipher is prohibitively high due to the large word widths (~ 512 bits). The use of redundant number representations amplifies this

problem by requiring multiple bits to represent each digit of the operands (e.g., the digit set {-1,0,1} requires 2 bits per digit), all of which must be registered on the chip. The algorithms used require numerous iterations as well. All of these factors add up to give a very large power dissipation, estimated using Powermill to be on the order of 938 mW at a supply voltage of 5V and a data rate of 1 Mbps. The following techniques can be utilized to minimize the power consumption of the QRG, in an effort to make it a feasible part of the low power encryption system.




(a) Schematic View

(b) Dependency Graph

**Figure 4-4:** Critical Path of Takagi's Multiplier

### 4.6.1 Conventional Supply Scaling

Analysis of the critical path of the QRG (Figure 4-4) revealed that the supply voltage must be kept at its maximum value of 5V in order to operate at the required clock rate of approximately 32 MHz that is required to meet the 1 Mbps throughput requirement.

### 4.6.2 Pipelining and Parallelizing the Computation

Unfortunately, the iterative nature of Takagi's algorithm, coupled with the dependency of each iteration on the results of the one before it, precludes the use of pipelining as a power reduction technique. Investigation of the algorithm also revealed that only the computation of the next $Y_j$ value could be performed in parallel with each iteration of the loop, resulting in a slight reduction in the critical path.

It should be noted that these deficiencies are not just a characteristic of Takagi's Algorithm. They are in fact a by-product of the basic approach utilized by these high-performance algorithms to perform fast modular multiplications.

### 4.6.3 Variable Supply Voltage

In many applications, the data rate is a time-varying function that is often less than the peak value (i.e., $\alpha = 1$). For example, in a video compression stream a majority of the time the throughput is much less than peak (Figure 4-5) due to the high correlation between video frames and the differential encoding algorithm used to compress the stream. In a conventional supply voltage scaling scheme the circuitry can be turned off using a gated clock once it has finished its computations. Thus the circuitry only operates a fraction of the time and hence the average power dissipation is given by the expression:

$$P_{avg} = \bar{\alpha} C V_{dd}^2 f \tag{4-9}$$

where $\bar{\alpha}$ is the average activity factor of the circuit. Table 4-1 shows the average activity factors and corresponding power reduction factors of three separate 400-frame compressed video sequences.

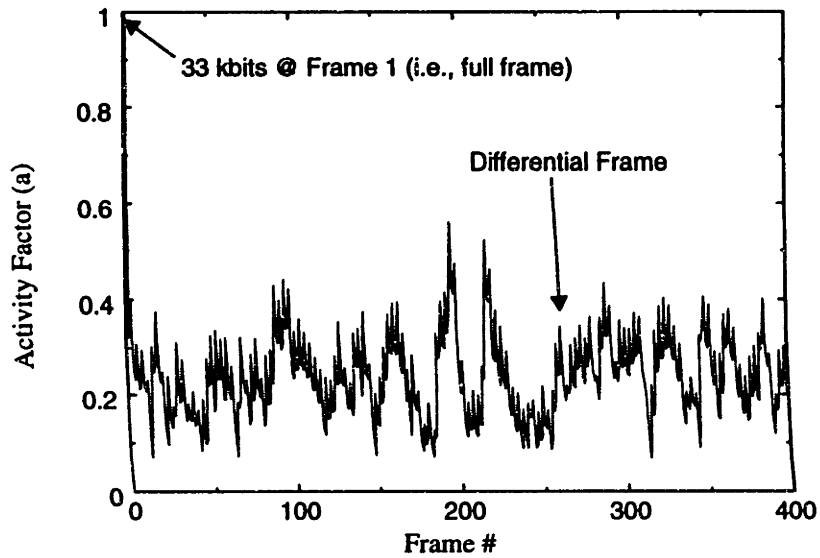



**Figure 4-5:** Activity Factor per Frame

| Sequence # | Average $\alpha$ ($= \bar{\alpha}$) | $P_{max}/P_{avg}$ ($= 1/\bar{\alpha}$) |
|---|---|---|
| 1 | 0.2400 | 4.17 |
| 2 | 0.1838 | 5.44 |
| 3 | 0.2529 | 3.95 |

**Table 4-1:** Activity Factors and Power Reduction Factors for 3 Video Sequences

The above linear reduction in power consumption can be improved upon by utilizing a variable supply voltage to reduce the supply voltage to the minimum required value, as determined by the activity factor, on a frame by frame basis. The basic idea is to lower the voltage when the activity is less than peak rather than working at a fixed supply and idling. By reducing the supply voltage, the power consumption can be reduced greater than quadratically (quadratically due to the reduction in $V_{dd}$, and linearly due to $\alpha$) as opposed to linearly when the circuit is allowed to idle.

The above technique was applied to the three compressed video streams. Figure 4-6 shows the resulting supply voltage for Sequence #1. Figure 4-7 shows the resulting power consumption for Sequence #1 utilizing a variable supply scheme, the power consumption of the fixed supply scheme is shown for reference. Table 4-2 shows the results of utilizing this method for all three video streams.

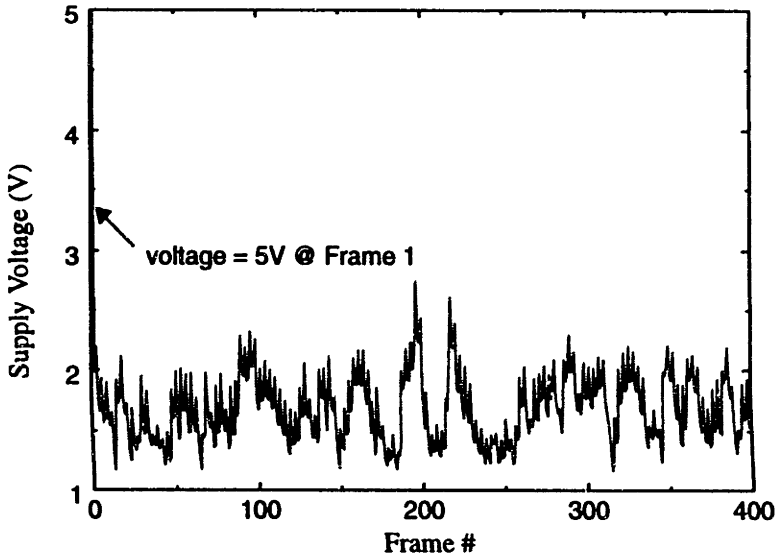



**Figure 4-6:** Supply Voltage per Frame

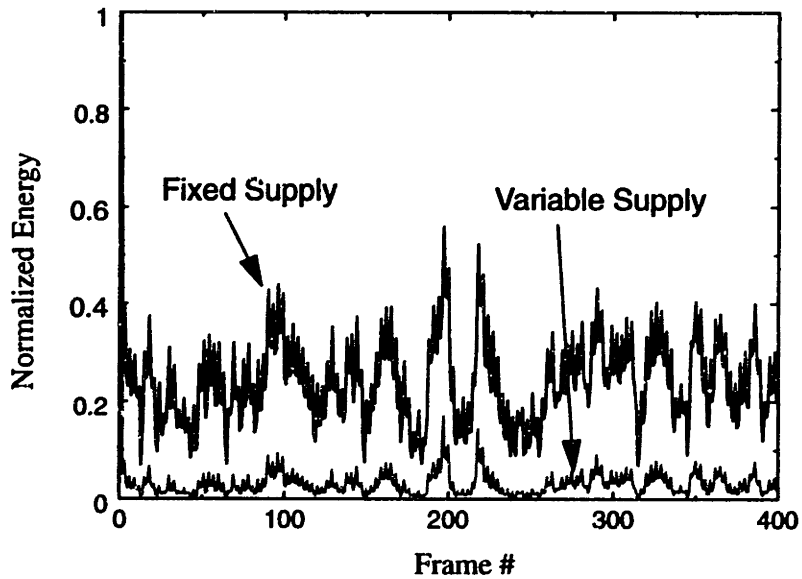



**Figure 4-7:** Normalized Energy Consumption per Frame

| Sequence # | $P_{variable\text{-}supply}/P_{fixed\text{-}supply}$ |
|---|---|
| 1 | 7.14 |
| 2 | 9.04 |
| 3 | 7.00 |

**Table 4-2:** Power Reduction Factor Using a Variable Supply Relative to a Fixed Supply

When the supply voltage is reduced the circuit delays increase (EQ 1-2), so the clock frequency must be reduced as well. One way to scale the clock proportionally with the voltage supply is to use a ring-oscillator connected to the varying supply voltage as the clock. The clock period will then track the supply voltage as the delays of the inverters in the ring oscillator will vary with the supply voltage. An initial implementation of this variable power supply and clock generator is described in [44].

### 4.6.4 Load Averaging

Further power reduction can be achieved by introducing buffering at the input of the encryption module and averaging the workload over multiple frames. It can be shown that by averaging the workload, the average power consumption is reduced based on the argument that the power consumption vs. throughput curve is convex and hence satisfies Jensen's Inequality (i.e., $E[f(x)] \geq f(E[x])$).

Averaging over multiple frames will increase the latency, but for bursty data patterns, such as differentially encoded video signals with frequent initialization frames (e.g., Figure 4-8), it will result in lower average power consumption. Figure 4-9 shows the normalized power consumption for a bursty 128 frame compressed video sequence utilizing the aforementioned variable voltage supply technique. Figure 4-10 shows the normalized amount of energy consumed per frame for varying averaging intervals. Figure 4-11 shows the normalized energy consumption as a function of the number of samples that are averaged for this video stream.

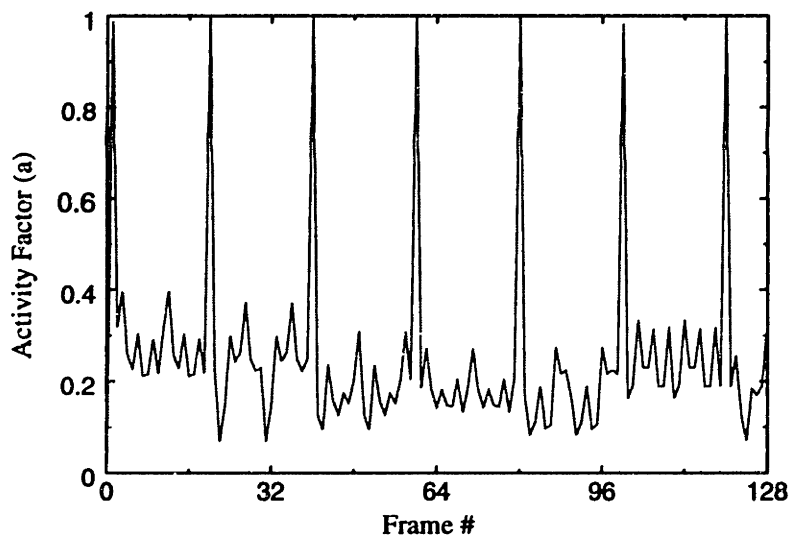



**Figure 4-8:** Activity Factor per Frame (bursty data)

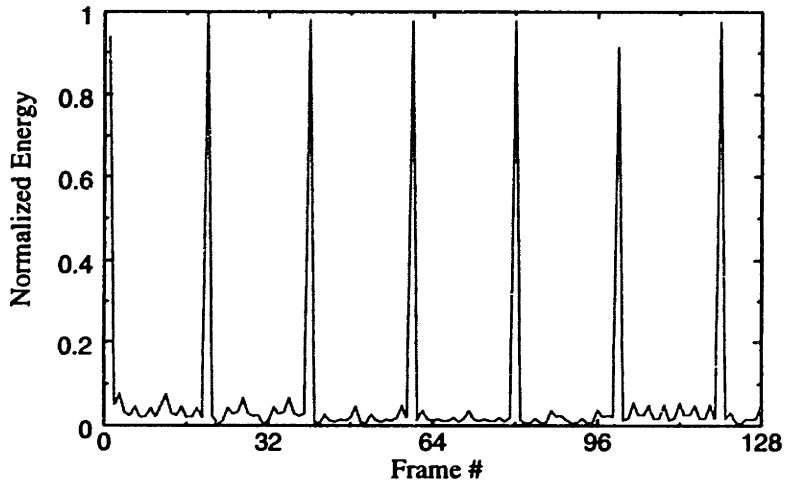



**Figure 4-9:** Energy Consumption per Frame (bursty data)

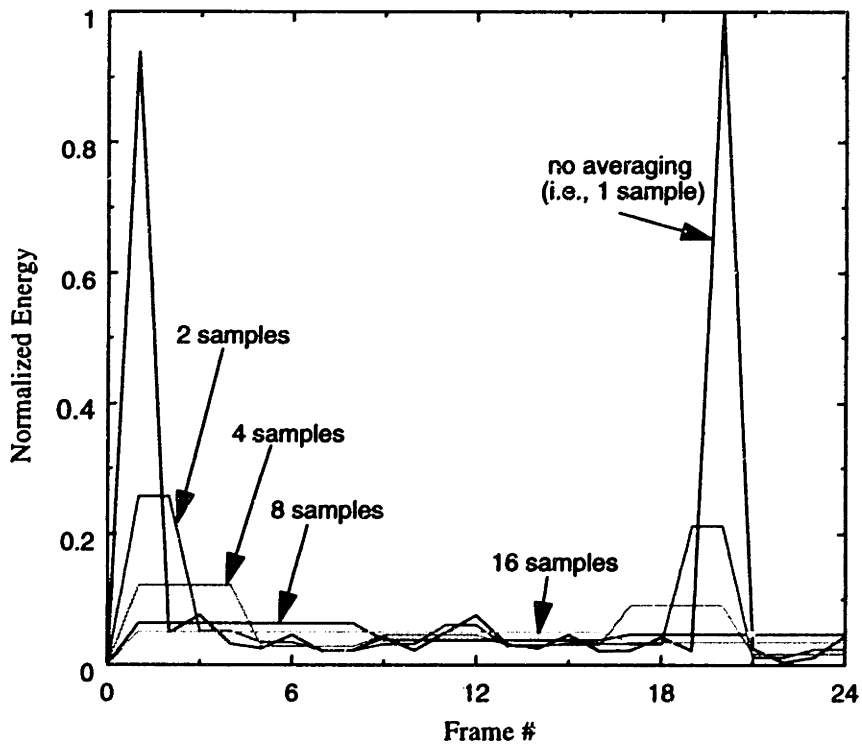



**Figure 4-10:** Energy Consumption per Frame for Varying Sample Sizes

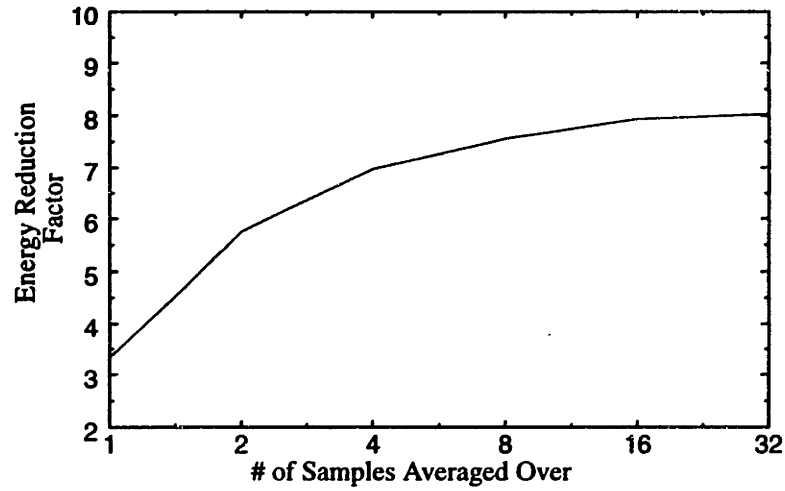



**Figure 4-11:** Energy Reduction Factor for Varying Sample Sizes Relative to a Fixed Supply Scheme

There is an inherent trade off between the variable voltage supply and averaging techniques. Variable voltage supply techniques favour data streams that are highly correlated (i.e., when the data rate is low with infrequent peaks such as in Figure 4-5), whereas averaging techniques favour bursty data streams (i.e., data rate has frequent peaks such as in Figure 4-8). Averaging techniques don't provide a significant amount of power reduction for non-bursty data streams (Table 4-3), while variable supply voltage techniques don't provide as noticeable an effect for bursty data streams (e.g., a reduction of 3.36 for the bursty data stream given in this section). However, when the two techniques are combined the power reduction factor approaches an order of magnitude for both types of data streams (Table 4-4). Hence the two techniques can be used in a complimentary fashion to provide significant power reduction across a wider range of data stream patterns.

| | Number of Frames per Sample | | | | |
|---|---|---|---|---|---|
| Sequence # | 1 | 2 | 4 | 8 | 16 |
| 1 | 7.14 | 7.65 | 7.86 | 8.06 | 8.36 |
| 2 | 9.05 | 9.88 | 10.09 | 10.17 | 10.38 |
| 3 | 7.00 | 7.39 | 7.53 | 7.66 | 7.75 |

**Table 4-3:** Power Reduction Factors Using Averaging Relative to a Fixed Supply Scheme

| Sequence # | $P_{var\&average}/P_{fixed}$ (averaged over 16 samples) |
|---|---|
| 1 | 8.36 |
| 2 | 10.38 |
| 3 | 7.75 |
| 4 (bursty data) | 7.91 |

**Table 4-4:** Power Reduction of Complimentary Scheme Relative to a Fixed Supply Scheme

## 4.7 Scalability

Many video compression algorithms generate a structured output containing both high priority and low priority information. For example, consider a differential encoding scheme in which the initial frame is transmitted uncompressed and then a sequence of difference frames are transmitted. In this example the initial frame would be labeled with a higher priority than the difference frames as without it the difference frames yield little useful information, whereas the differential frames could be approximated using interpolation. The system designer can utilize their knowledge of the video compression algorithm to dynamically reconfigure the encryption module to provide varying levels of security for the data stream based on the priorities of the data being transmitted. This is similar to the idea of priority encoding (e.g., [45]) that is used to allocate additional error recovery coding for portions of the data stream that are deemed important (i.e, high priority), and reduced error correction coding for the lower priority portions of the data stream.

The security of the QRG scales with the size of the modulus and hence the width of the modular multiplier, which can be can be dynamically scaled to meet the varying security requirements of the data stream. The power consumption of the multiplier varies approximately inversely with the multiplier width (Figure 4-12), so relatively small reductions in the modulus width correspond to large reductions in the power consumption.

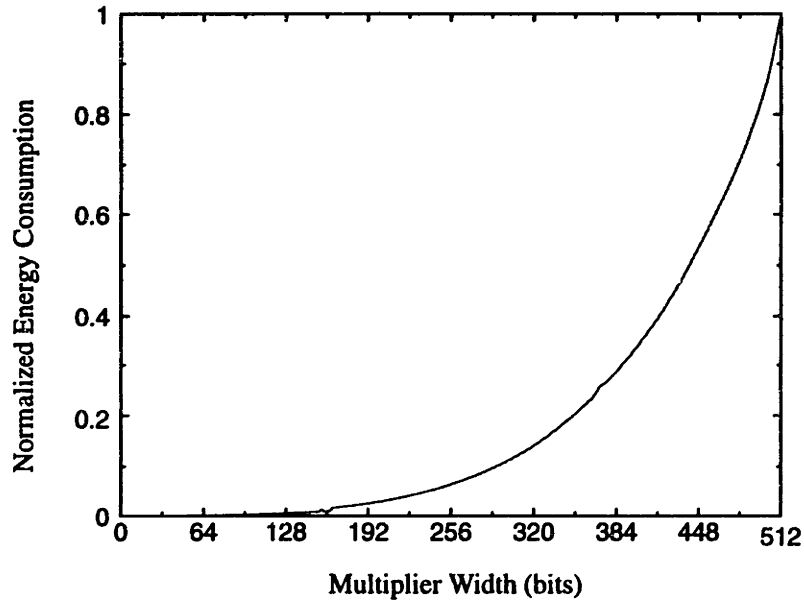



**Figure 4-12:** Energy Consumption for Varying Multiplier Widths

The security of the LFSR-based stream ciphers scales with the lengths of the LFSRs used. The register lengths can also be dynamically scaled to provide varying levels of encryption. The power consumption of the cipher scales linearly with register length (Figure 3-6), hence the power consumption can be reduced by approximately a linear amount.

As an example, video sequence #1 from Section 4.6.3 was encrypted using the QRG following the priority scheme given in Table 4-5. An average power reduction factor on the order of 2.8 relative to a constant-priority scheme was obtained using the scalable technique. Repeating the experiment using the Shrinking Generator as the stream cipher yielded a power reduction on the order of 1.8.

| | | QRG | | Shrinking Generator | |
|---|---|---|---|---|---|
| Frame Type | Priority | Modulus Width (bits) | Normalized Power Consumption | LFSR Widths (bits) | Normalized Power Consumption |
| Intra-frame (single frame) | HIGH | 512 | 1.000 | 78 | 1.00 |
| Inter-frame (differential frame) | LOW | 384 | 0.288 | 39 | 0.500 |

**Table 4-5:** Priority assignment for scalability example

## 4.8 Test Chip Design

At the topmost level the modular multiplier test chip consists of five separate blocks (Figure 4-13). The Bitslice Core, $C_j$-Selector and Y-Recoder form the datapath of the multiplier and operate under the control of the Controller block. The Serial I/O block serves as a test port through which the multiplier can be initialized for testing purposes.

A full set of schematics of the Modular Multiplier Test Chip can be found in Appendix B.

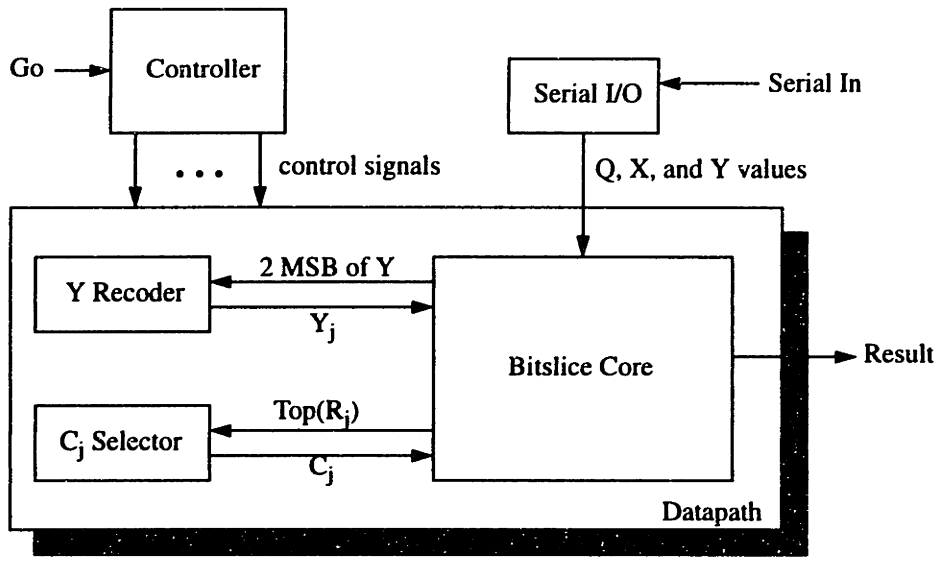



**Figure 4-13:** Modular Multiplier Test Chip Top Level Block Diagram

### Bitslice Core

The Bitslice Core implements two digit selections and two redundant binary additions (Figure 4-14). The value of $Y_j$ is first used to select a multiple of X ($\pm X$ or $\pm 2X$) as the input to the first redundant adder where it is added to the previous intermediate result ($P_{j+1}$). The result of this addition ($R_j$) is used to compute the quotient estimate ($C_j$) and that value is used to select a multiple of Q ($\pm 4Q$ or $\pm 8Q$) that is then used to partially reduce the intermediate result via the second redundant adder. The output of the second redundant adder is then stored as the new intermediate result ($P_j$).

An *n*-bit implementation of Takagi's Algorithm requires a total of *n*+3 bitslices in the core, the most significant of which is a special case used to adjust the most significant two digits of $P_j$ to ensure that the intermediate result can be stored in just *n*+2 digits. The

Bitslice Core represents the majority of the logic and device count within the multiplier and thus the decision was made to do a full-custom layout of the bitslice to facilitate an area-efficient layout of the multiplier. The final bitslice contained a total of 234 devices and occupied an area of 19,427 $\mu m^2$.

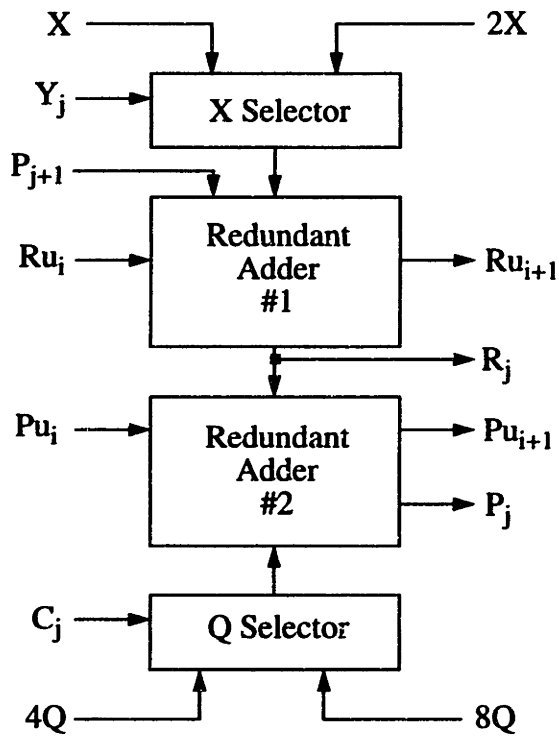



**Figure 4-14:** Bitslice Core Block Diagram

**Y Recoder**

The Y Recoder scans the Y operand of the multiplier two bits at a time from most significant to least significant, recoding each pair into a single minimally-redundant radix-4 value ($Y_j$) whose value can be any of {-2, -1, 0, 1, 2}. The calculation of the next $Y_j$ value is pipelined so that it can be overlapped with the current iteration to reduce the cycle time of the multiplier (Figure 4-15).

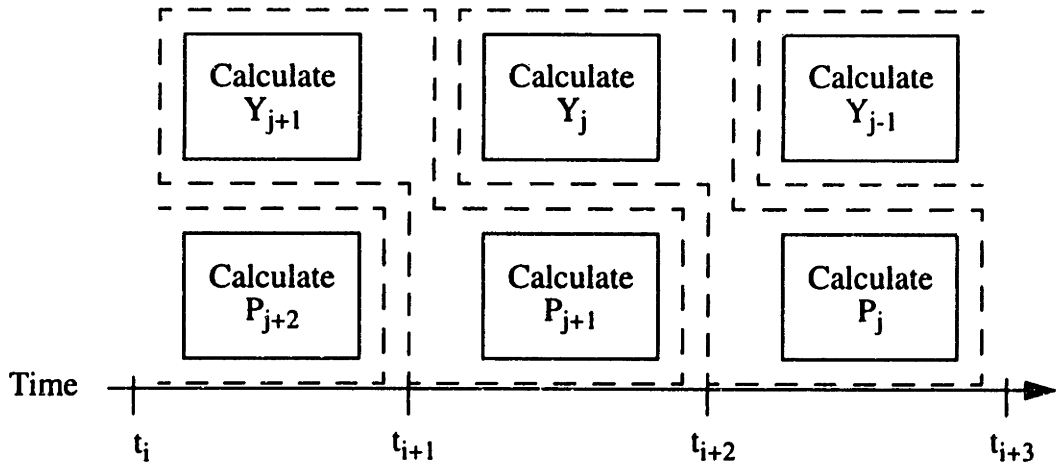



**Figure 4-15:** Pipeline Flow of Y Recoder

### $C_j$ Selector

The $C_j$ Selector (Figure 4-16) is responsible for calculating the approximate quotient digit on each iteration of the multiplier. The Selector operates according to the following rule:

$$C_j = \begin{cases} -2, \text{ if } (top(R_j) < -top(6 \cdot Q)) \\ -1, \text{ if } (-top(6 \cdot Q) \le top(R_j) < -top(2 \cdot Q)) \\ 0, \text{ if } (top(2 \cdot Q) \le top(R_j) < top(2 \cdot Q)) \\ 1, \text{ if } (top(2 \cdot Q) \le top(R_j) < top(6 \cdot Q)) \\ 2, \text{ if } (top(R_j) \ge top(6 \cdot Q)) \end{cases} \quad (4\text{-}10)$$

where $top(R_j)$ is the most significant 8 digits of $R_j$, $top(2Q)$ is the most significant 5 digits of 2Q and $top(6Q)$ is calculated from the most significant 6 digits of $Q$.

The calculation of $C_j$ begins by converting $top(R_j)$ to a two's complement non-redundant binary form using an addition. If the non-redundant form of $top(R_j)$ is positive, it is inverted and then subtracted from 2Q; if it is negative, it is added. The result is then subtracted from 4Q if it is positive, and added if it is negative. The final addition is actually a carry-out computation as only the sign of the last addition is required to calculate $C_j$. The resulting sign bits from each addition/subtraction are decoded to compute the value of $C_j$

(Table 4-6). This method always attempts to minimize the absolute value of the sum at each stage. This in turn minimizes the size of the additions and hence the delay of the $C_j$ calculation.

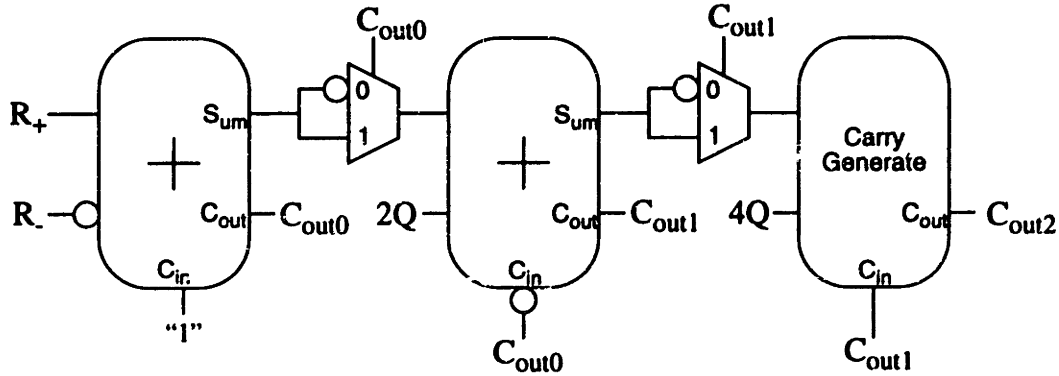



**Figure 4-16:** $C_j$ Selector Block Diagram

| $C_{out0}$ | $C_{out1}$ | $C_{out2}$ | $C_j$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 2 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | -1 |
| 1 | 1 | 1 | -2 |

**Table 4-6:** Decoding of Carry-out Bits in $C_j$ Selector

**Controller**

The Controller generates all of the control signals required for operating the multiplier. The Controller was designed with the intention of being completely scalable to allow for varying multiplier widths. This scalability is accomplished by partitioning the controller state machine such that there is an initial setup phase, then an iterative phase which is performed as many times as required by the width of the multiplier operands and finally a fin-

ishing phase that completes the multiplication (Figure 4-17). Both the initial setup and finishing phases are generic and applicable to all operand sizes, whereas the iterative phase need only be repeated $n/2$ times for $n$-bit operands

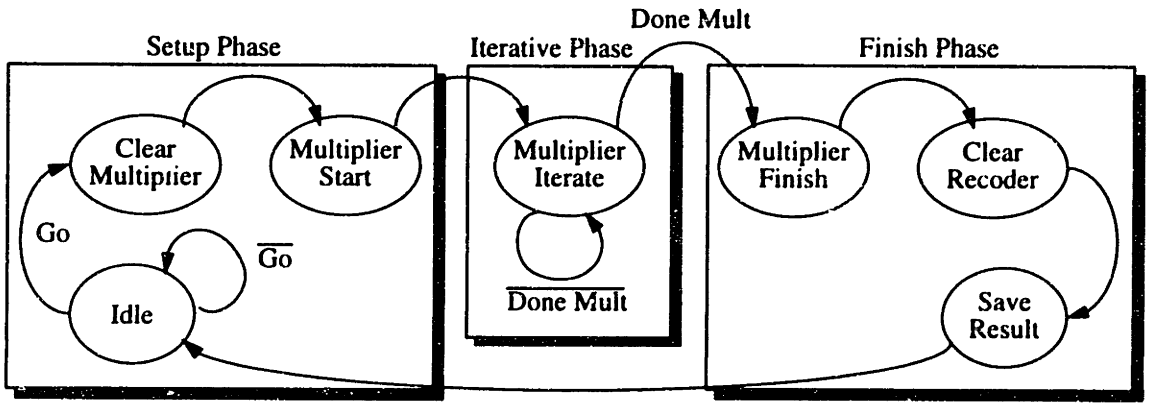



**Figure 4-17:** Controller State Diagram

### Design for Scalability

The modular multiplier architecture was designed to be fully scalable in a real implementation, hence the design choices made in the Controller logic. Takagi's Algorithm is particularly well suited to a scalable implementation. Given an $n$-bit implementation, Takagi's Algorithm can be scaled to perform $m \leq n$ modular multiplications with minimal modifications to the datapath and controller logic. The $m$-bit operands must be loaded into registers using the most significant $m$-bits of the register. The unused low order bits can be turned off by disabling their clocks (this will effectively disable the least significant ($n$ - $m$) bit slices as well). The algorithm is then iterated a total of $m/2$ times to generate the $m$-bit result. The controller can utilize a ($\log_2 n$ - 1) bit counter, ($\log_2 n$ - 1) bit comparator and a size register to create the necessary control information that ensures the multiplier iterates for the required number of cycles.

### Layout

The Modular Multiplier Testchip was implemented in a 0.6 µm process, requiring a total of 5963 devices, and an area of 1,163,530 $\mu m^2$. Figure 4-18 shows the layout of the testchip.

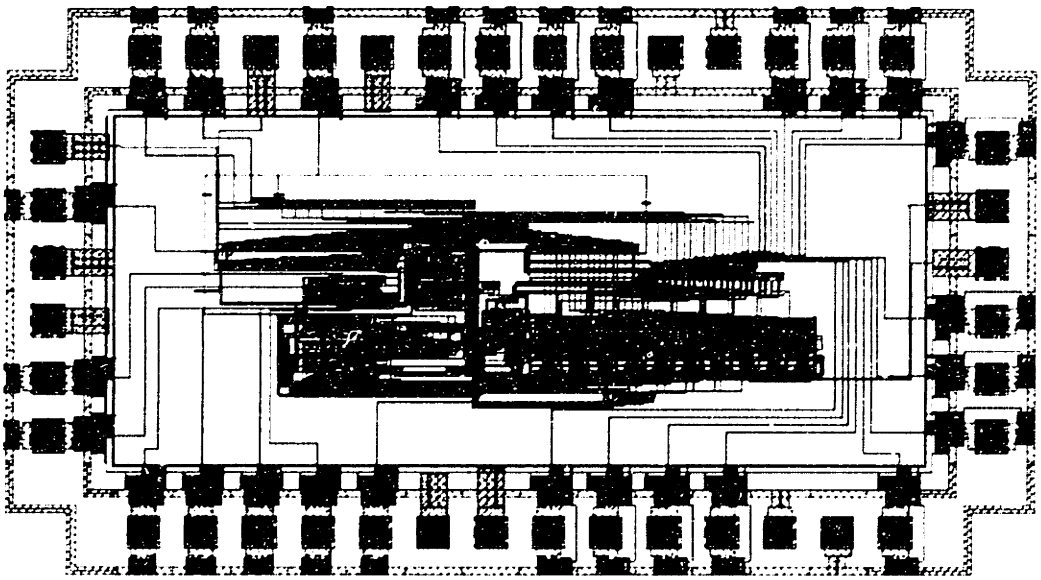


**Figure 4-18:** Modular Multiplier Test Chip Layout

## 4.9 Preliminary Testing

Preliminary power consumption estimates were determined from the extracted layout of the testchip using Powermill. Powermill predicted that the power dissipation would be 4.58 mW at a supply voltage of 5V and a clock frequency of 10 MHz. Measurements from the testchip showed this number to be pessimistic, the power consumption of the actual chip using the same operating conditions and stimuli was just 4.16 mW. Estimates of the power consumption for a full scale 512-bit multiplier are shown Figure 4-19. These estimates were calculated by first determining the equivalent switched capacitance of the 8-bit multiplier and then assuming that a 512-bit implementation would require 64 copies of the 8-bit version. Given these assumptions, the power consumption of the 512-bit multiplier can be calculated using the following expression:

$$P_{512} = \left( \frac{64 \cdot I_8}{V_8 \cdot f_8} \right) \cdot V_{512}^2 \cdot f_{512} \qquad (4\text{-}11)$$

where $V_8$, $I_8$, are $f_8$ taken from the test measurement and setup, $f_{512}$ = 32 MHz, and $V_{512}$

= $V_8$. Note that this calculated value will be pessimistic in the sense that it includes the power consumption of the control logic, $Y_j$ recoder and $C_j$ selector in each 8-bit slice. In a real implementation, the power consumption of these blocks will be amortized over many more bitslices.

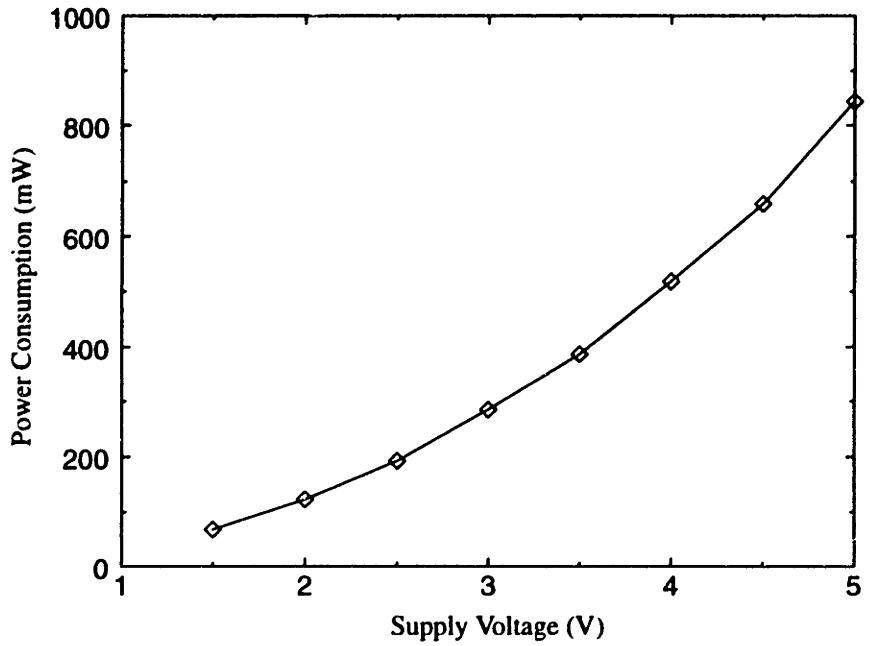



**Figure 4-19:** Estimated Power Consumption of QRG Based on 8-bit Test Chip

Measurements were performed to determine the maximum operating speed of test chip over a range of supply voltages. The chip was stimulated with the maximum frequency possible using the test setup (i.e., $f_{mult}$ = 12.5 MHz) and the supply voltage was reduced until the multiplier began to fail. The equivalent delay factor was then calculated for a range of supply voltages from the process data used to construct Figure 1-2. From these values an equivalent maximum clock frequency could be derived for each supply voltage, along with the throughput (Table 4-7). Using this technique, the maximum operating frequency of the test chip was found to be approximately 38 MHz, exceeding our requirement of 32 MHz by over 18%. This allows us to operate the modular multiplier at a reduced supply voltage of approximately 4V and still satisfy the 1Mbps throughput requirements of our application, yielding a power reduction of 36%.

| Supply Voltage (V) | Delay Factor | Equivalent Maximum Frequency (MHz) | Throughput (Mbps) | Power Consumption (mW) |
|---|---|---|---|---|
| 5.0 V | 0.330 | 37.93 MHz | 1.18 | 1000.2 |
| 4.5 V | 0.351 | 35.58 MHz | 1.11 | 733.7 |
| 4.0 V | 0.393 | 31.79 MHz | 0.99 | 514.2 |
| 3.5 V | 0.455 | 27.48 MHz | 0.85 | 332.4 |
| 3.0 V | 0.531 | 23.55 MHz | 0.73 | 209.8 |
| 2.5 V | 0.670 | 18.65 MHz | 0.58 | 112.2 |
| 2.0 V | 1 | 12.5 MHz | 0.39 | 48.0 |
| 1.5 V | 1.675 | 7.46 MHz | 0.23 | 15.8 |

**Table 4-7:** Test Data for Maximum Clock Frequency Based on Measurements at $V_{dd} = 2V$

# Chapter 5

# Conclusions and Future Work

The high BERs that are inherent in wireless communication channels, coupled with widely varying data rates make stream ciphers the algorithm of choice for data encryption.

LFSR-based stream ciphers are a very attractive choice of encryption algorithm as they feature very simple hardware requirements that are very efficient for providing high-speed, low-power keystream generation. On the other hand, the QRG provides much more security at the cost of a much more complex algorithm that consumes considerably more power than the simpler, less secure LFSR-based stream cipher.

A dynamically scalable encryption scheme can be utilized to provide more secure encryption for high priority data, and less encryption for low priority data in those applications where the data stream can be explicitly prioritized. By scaling the encryption the power consumption of the system can be significantly reduced (e.g., 2 - 3 times less power) over a non-prioritized implementation. All of the LFSR-based stream cipher implementations and the QRG based on Takagi's modular multiplying algorithm described herein feature scalable architectures for implementing a scalable scheme.

The use of a dynamic voltage supply allows for significant power reduction in applications where the computational requirements vary over time. If additional latency can be tolerated in the signal path then averaging techniques can be utilized (at the cost of additional buffering) to provide further reductions in power, especially for those applications where the variance of the computational requirements is high. Averaging and variable supply techniques act in a complimentary manner to yield roughly an order of magnitude reduction in power consumption in cases where either of the techniques by themselves yields little improvement (i.e., more improvement over a wider range of data types).

While the power consumption of the QRG (843 mW) may be low enough for some applications, it is prohibitively high for ultra low power applications where power budgets

are on the order of 50mW. Hence the QRG should not be used as a keystream generator. However, its cryptographically secure output would serve as an excellent pseudo-random key generator for a low power LFSR-based stream cipher. If used in such a role the generator could operate in the background at a much lower data rate and supply voltage, thereby making it a feasible component for the low-power encryption module. Other benefits of the hybrid system are that it helps minimize the effects of synchronization loss that plagues both stream and block ciphers and it increases the amount of work that an attacker must do to attack the system by forcing them to repeatedly crack the encryption every time the key is regenerated.

As an example, consider a 1Mbps keystream that is reinitialized every 10 frames (i.e., 300 ms), requiring approximately 100 bits per initialization. The key generator data rate is 100 bits / 300 ms = 333 bps, corresponding to an allowable delay increase by a factor of 1Mbps/333 bps = 3000. At a supply voltage of 1V circuit delays increase by a factor of 40 so the clock frequency can be brought down to 32MHz / 40 = 800 kHz. At a 800 kHz clock frequency the QRG will output 800 kHz / 256 (cycles / multiply) × 8 (bits / multiply) = 25 kbps. Thus the generator need only operate 1/75 of the time and can either be turned off the remainder of the time or the supply voltage can be lowered even further. Assuming that the generator is turned off the power reduction factor of the hybrid implementation is 3000x, for an estimated power consumption 450.0 μW. When coupled with a low power LFSR-based stream cipher, such as the ASG, the total power consumption of a hybrid system will be less than 1/2 mW.

## 5.1 Future Work

The body of work described within this thesis is only the beginning in terms of what must be done for developing a secure wireless communications link. The following are several research projects that should be investigated as a continuation of the work previously described herein.

### 5.1.1 Low Power Hybrid System

The speed and efficiency of the LFSR-based stream cipher can be combined with the security of QRG-based stream ciphers to yield a low power hybrid system (Figure 5-1). A simple, low-power LFSR-based stream cipher is used to generate the output keystream, which

is decoupled from the data stream through an output buffer and the more complex, and power intensive, QRG is used as the seed generator. At the time of re-synchronization, the LFSR-based cipher is re-initialized with the pseudo-random bits in the seed buffer and the feedback polynomial registers are loaded with values from a low-power polynomial ROM that are selected using several pseudo-random bits from the seed buffer.

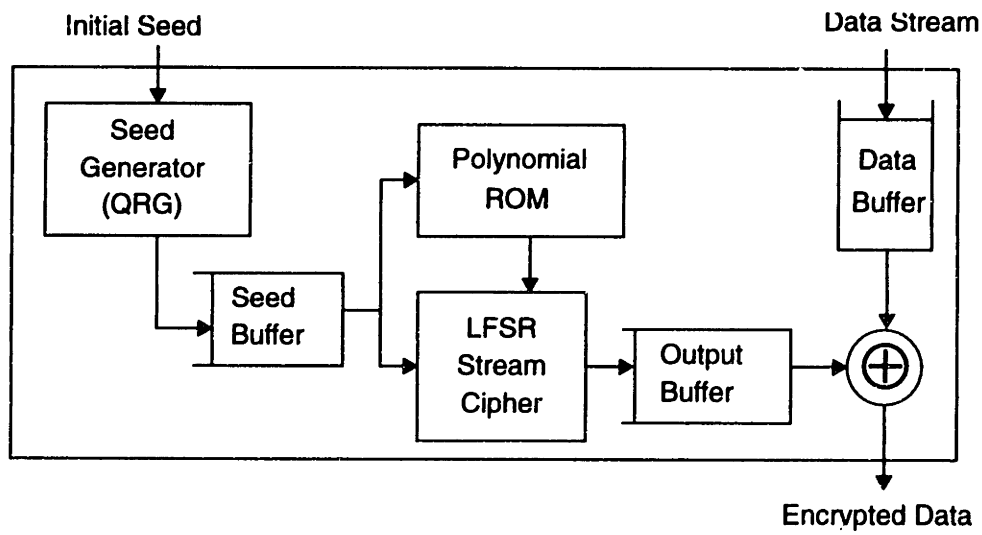



**Figure 5-1:** Proposed Low Power Encryption System

Further work needs to be done to develop both the major functional blocks (i.e., a 512+ bit QRG and programmable LFSR Stream Cipher) and support circuitry (i.e., low power ROMs and FIFOs/buffers) of such a system if it is to be used in an ultra low power application.

### 5.1.2 Key Exchange and Authentication

The initial seeds of the QRG can be hardwired into both the sensor unit and the base station, much like the unique identification key of the GSM standard. Future work should be done to determine the feasibility of low power key exchange and authentication protocols for use in wireless networks. The work should occur at both a high-level transaction level (e.g., minimizing the number of computations required while still ensuring a tamper-proof transaction) and at the low-level implementation level (e.g., developing low-power implementations of the necessary protocol building blocks such as hash functions and comparators).

### 5.1.3 Full-scale Modular Multiplier Chip

Currently we are only able to estimate the power consumption of a full-scale 512-bit quadratic residue generator, based on measurements obtained from an 8-bit implementation. The next step is to build a full-scale modular multiplier chip which will be used to verify these estimates and will serve as a piece of the proposed low-power hybrid encryption system of Section 5.2.1.

### 5.1.4 Modular Multiplication Algorithm Development and Implementation

The performance of QRG depends almost completely on the ability to perform repeated modular squarings quickly and efficiently. Hence work should be done to develop algorithms for performing modular squarings quickly and efficiently. Perhaps the fact that we are performing squarings rather than more general multiplication can be exploited to yield significant gains in performance[4]. In addition, extending existing algorithms, such as Takagi's, to higher-radix implementations should be investigated as the running time of these algorithms is directly proportional to $\log_2(\text{radix})$. So by extending the algorithm to radix 8, 16 or even 32, we can get improvements on the order of 1.5x, 2x, and 2.5x repsectively in terms of performance. However, using a higher-radix will require additional overhead in terms of additional recoding, more complex redundant addition, increased storage requirements and more complex quotient estimation. It remains to be seen at what point the benefits of going to an even higher radix are outweighed by the costs.

4. Note that though we might make significant gains (e.g., 2x or 4x), both squaring and multiplication are algorithmically equivalent in terms of running time as $XY = ((X + Y)^2 - (X - Y)^2) / 4$, hence any algorithmic breakthrough that would allow us to perform modular squarings could be exploited to perform modular multiplications as well.

# References


[1] J. Eckhouse, "Hackers Hurt Cellular Industry," *San Francisco Chronicle*, January 25, 1993, page C1.

[2] Cellular Telecommunications Industry Association, "National Phone Fraud Expert Testifies In Favour of Maryland State Police Proposal," *CTIA Press Release*, January 31, 1996

[3] A.P. Chandrakasan, *Low Power Digital CMOS Design*, Ph. D. Thesis, UC Berkeley, 1994.

[4] W. Diffie, M.E. Hellman, "New Directions in Cryptography," *IEEE Transactions on Information Theory*, vol. IT-22, no. 6, November 1976, pp. 644-654

[5] T. ElGamal, "A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms," *Advances in Cryptology - CRYPTO '84 Proceedings*, Springer-Verlag, 1985, pp. 10-18.

[6] R.L. Rivest, A. Shamir, L.M. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," *Communications of the ACM*, vol. 21, no. 2, February 1978, pp. 120-126.

[7] A.M. Odlyzko, "The Future of Integer Factorization," *CryptoBytes*, RSA Laboratories, vol. 1, no. 2, Summer, 1995, pp. 5-12.

[8] H. Lin, L. Harn, "Authentication in Wireless Communications," *GLOBECOM '93*, 1993, pp. 550-554.

[9] A. Aziz, W. Diffie, "Privacy and Authentication for Wireless Local Area Networks," IEEE Personal Communications, vol. 1, no. 1, 1994, pp. 25-31.

[10] National Institute of Standards and Technology, "Data Encryption Standard," *NIST FIPS PUB 46-2*, U.S. Department of Commerce, December 1993.

[11] X. Lai, *On the Design and Security of Block Ciphers*, ETH Series in Information Processing, vol. 1, Konstanz: Hartung-Gorre Verlag, 1992.

[12] B. Schneier, "The Blowfish Encryption Algorithm," *Dr. Dobb's Journal*, vol. 19, no. 4, April 1994, pp. 38-40.

[13] R.L. Rivest, "The RC5 Encryption Algorithm," *Dr. Dobb's Journal*, vol. 20, no. 1, January 1995, pp. 146-148.

[14] L.R. Knudsen, *Block Ciphers - Analysis, Design, Applications*, Ph. D. dissertation, Aarhus University, November, 1994.

[15] R.A. Rueppel, *Analysis and Design of Stream Ciphers*, Springer-Verlag, 1986

[16] J.L. Massey, "Shift-Register Synthesis and BCH Decoding," *IEEE Transactions on Information Theory*, vol. IT-15, no. 1, January 1969, pp. 122-127

[17] L. Blum, M. Blum, M. Shub, "A Simple Unpredictable Pseudo-Random Number Generator," *SIAM Journal on Computing*, vol. 15, no. 2, 1986, pp. 364-383.

[18] T. Meng, B.M. Gordon, E.K. Tsern, A.C. Hung, "Portable Video-on-Demand in Wireless Communication," *Proceedings of the IEEE, vol. 83, no. 4, April 1995, pp. 659 - 680.*

[19] D.A. Huffman, "A method for the construction of minimum-redundancy code," *Proc. IRE*, vol. 40, September 1952, pp. 1098-1101.

[20] A.K. Lenstra, H.W. Lenstra, Jr., M.S. Manasse, J.M. Pollard, "The Number Field Sieve," *Proceedings of the Twenty Second Annual ACM Symposium on Theory of Computing*, 1990, pp. 564-572.

[21] C. Pomerance, "The Quadratic Sieve Factoring Algorithm," *Advances in Cryptology -- Proceedings of EUROCRYPT '84*, Springer-Verlag, 1985, pp.169-182.

[22] European Telecommunications Standards Institute, "Security Aspects," *Recommendation GSM 02.09.*

[23] European Telecommunications Standards Institute, "Subscriber Identity Module," *Recommendation GSM 02.17.*

[24] European Telecommunications Standards Institute, "Security Related Network Functions," *Recommendation GSM 03.20.*

[25] T. Beth, F.C. Piper, "The Stop-and-Go Generator," *Advances in Cryptology - EUROCRYPT '84 Proceedings*, Springer-Verlag, 1984, pp. 88-92.

[26] P.R. Geffe, "How to Protect Data With Ciphers That are Really Hard to Break," Electronics, vol. 46, no. 1, January 1973, pp. 99-101.

[27] S.M. Jennings, *A Special Class of Binary Sequences*, Ph.D. Dissertation, University of London, 1980.

[28] D. Coppersmith, H. Krawczyk, Y. Masour, "The Shrinking Generator," *Advances in Cryptology - CRYPTO '93 Proceedings*, Springer-Verlag, 1995, pp. 22-39.

[29] I. Kessler, H. Krawczyk, "Minimum Buffer Length and Clock Rate for the Shrinking Generator Cryptosystem," *IBM Research Report RC 19938(88322)*, IBM Research Division, February 17, 1995

[30] J.D. Golic, "Linear Cryptanalysis of Stream Ciphers," *Fast Software Encryption - Second International Workshop Proceedings*, Springer-Verlag, 1995, pp. 154-169.

[31] W. Meier, O. Staffelbach, "The Self-Shrinking Generator," *Advances in Cryptology - EUROCRYPT '94 Proceedings*, Springer-Verlag, 1995, pp. 205-214.

[32] C.G. Gunther, "Alternating Step Generators Controlled by de Bruijn Sequences," *Advances in Cryptology -- EUROCRYPT '87 Proceedings*, Springer-Verlag, 1988, pp. 5-14.

[33] B. Schneier, *Applied Cryptography - Second Edition*, New York: John Wilely & Sons Inc., 1996, pp. 389, 662-667.

[34] R.J. Anderson, Posting to sci.crypt USENET newsgroup, June 17, 1994

[35] T. Barber, *BodyLAN: A Low Power Communications System*, M.S. Thesis, MIT, 1996, pp. x

[36] D. Stinson, *Cryptography: Theory and Practice*, Boca Raton: CRC Press Inc., 1995.

[37] U.V. Vazirani, V.V. Vazirani, "Efficient and Secure Pseudo-Random Number Generation," *Advances in Cryptology - CRYPTO '84 Proceedings*, Springer-Verlag, 1985, pp. 193-202.

[38] H. Makino, Y. Nakase, H. Suzuki, H. Morinaka, H. Shinohara, K. Mashiko, "An 8.8-ns 54*54-bit Multiplier With High Speed Redundant Binary Architecture," *IEEE Journal of Solid-State Circuits*, vol. 31, no. 6, June 1996, pp. 773-783.

[39] R.K. Yu, G.B. Zyner, "167 MHz Radix-4 Floating Point Multiplier," *Proceedings of the 12th Symposium on Computer Arithmetic*, 1995, pp. 149-154

[40] N. Ohkubo, M. Suzuki, T. Shinbo, T. Yamanaka, A. Shimizu, K. Sasaki, Y. Nakagome, "A 4.4-ns CMOS 54*54-b Multiplier Using Pass-Transistor Multiplexer," *Proceedings of IEEE Custom Integrated Circuits Conference - CICC '94*, 1994, pp. 599-602.

[41] N. Takagi, "A Radix-4 Modular Multiplication Hardware Algorithm for Modular Exponentiation," *IEEE Transactions on Computers*, vol. 41, no. 8., August 1992, pp. 949-956.

[42] H. Morita, "A Fast Modular-Multiplication Algorithm Based on a Higher Radix," *Advances in Cryptology - CRYPTO '89 Proceedings*, Springer-Verlag, 1990, pp. 387-399.

[43] H. Orup, P. Kornerup, "A High-Radix Hardware Algorithm for Calculating the Exponential $M^E$ Modulo N," *10th IEEE Symposium on Computer Arithmetic*, 1991, pp. 51-57.

[44] V. Gutnik, *Variable Supply-Voltage for Low Power DSP*. M.S. Thesis, MIT, 1996.

[45] A. Albanese, J. Blomer, J. Edmonds, M. Luby, M. Sudan, "Priority Encoding Transmission," *Proceedings of 35th FOCS*, 1994, pp. 604-612.

# Appendix A

# Stream Cipher Schematics

This section provides schematics for the major functional blocks of the Stream Cipher Test Chip. A list of provided schematics is given below along with a brief description of each.

- AltStopGoGenerator1 - alternating stop & go generator stream cipher
- ShrinkingGenerator1 - shrinking generator stream cipher
- SelfShrinkGenerator1 - self-shrinking generator stream cipher
- A5 - A5 stream cipher
- LFSR8x8+1 - 65 bit variable length programmable linear feedback shift register
- LFSR8x8 - 64 bit variable length programmable linear feedback shift register
- LFSR8x8-1 - 63 bit variable length programmable linear feedback shift register

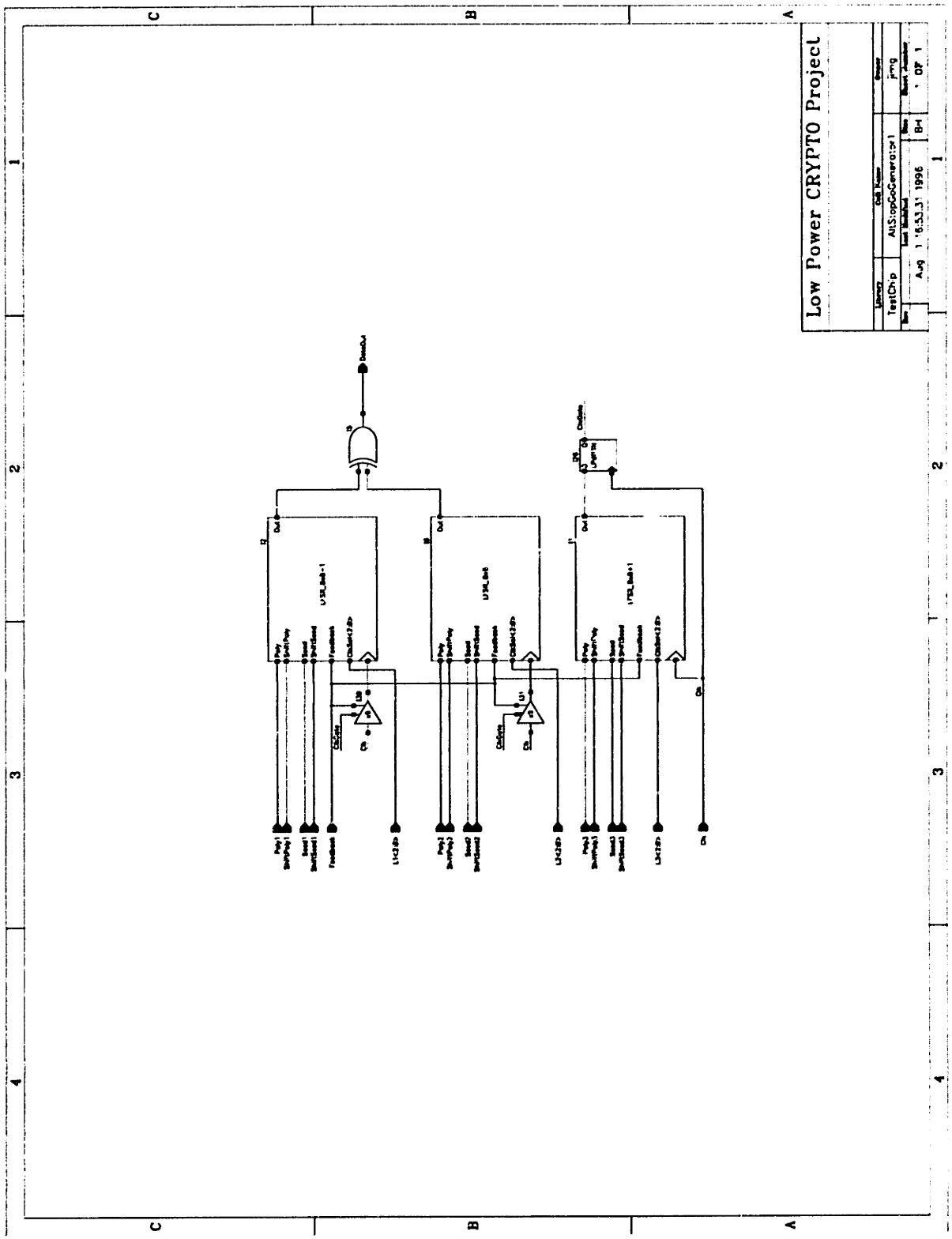



**Figure A-1:** Alternating Stop & Go Generator Schematic

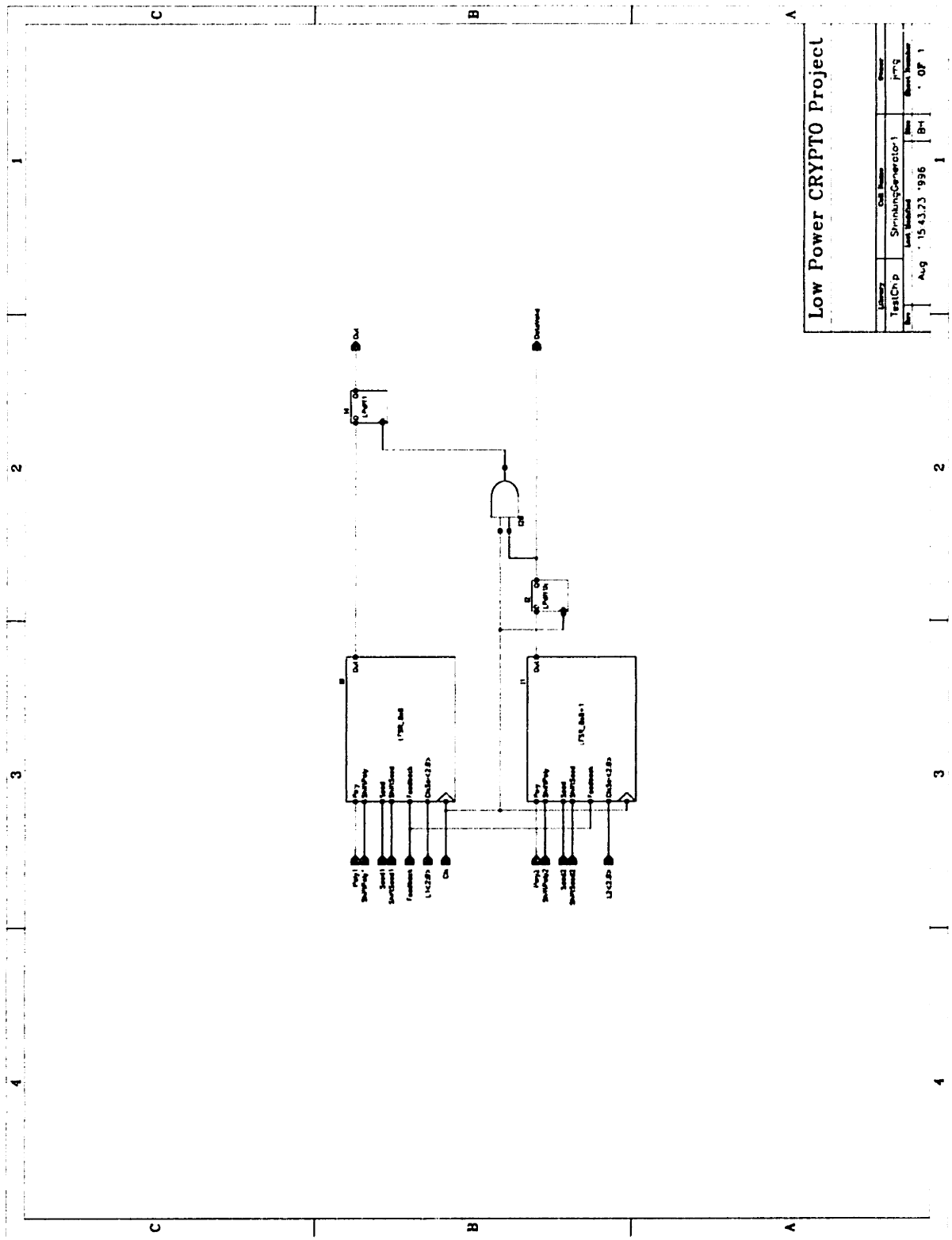



**Figure A-2:** Shrinking Generator Schematic

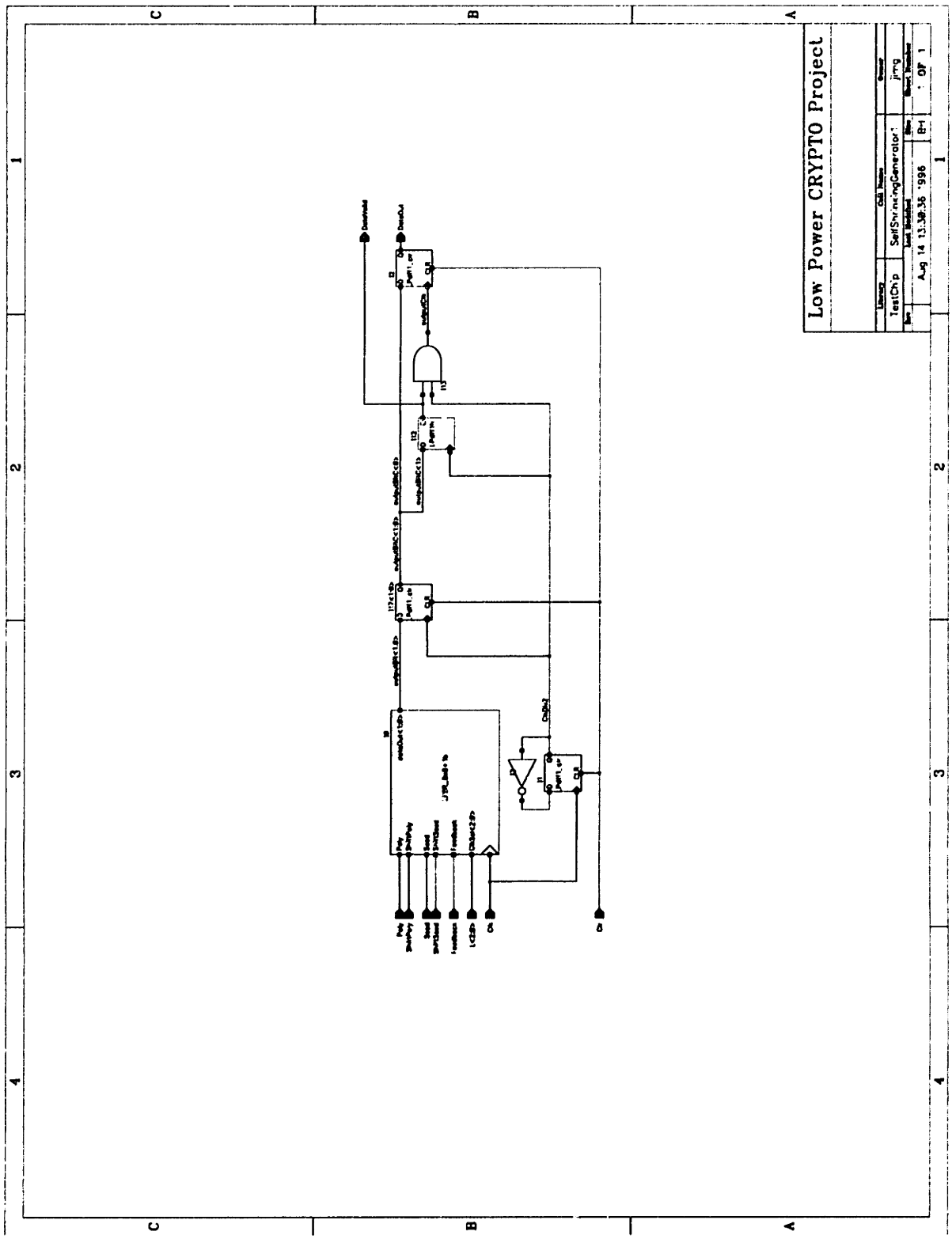



**Figure A-3:** Self-Shrinking Generator Schematic

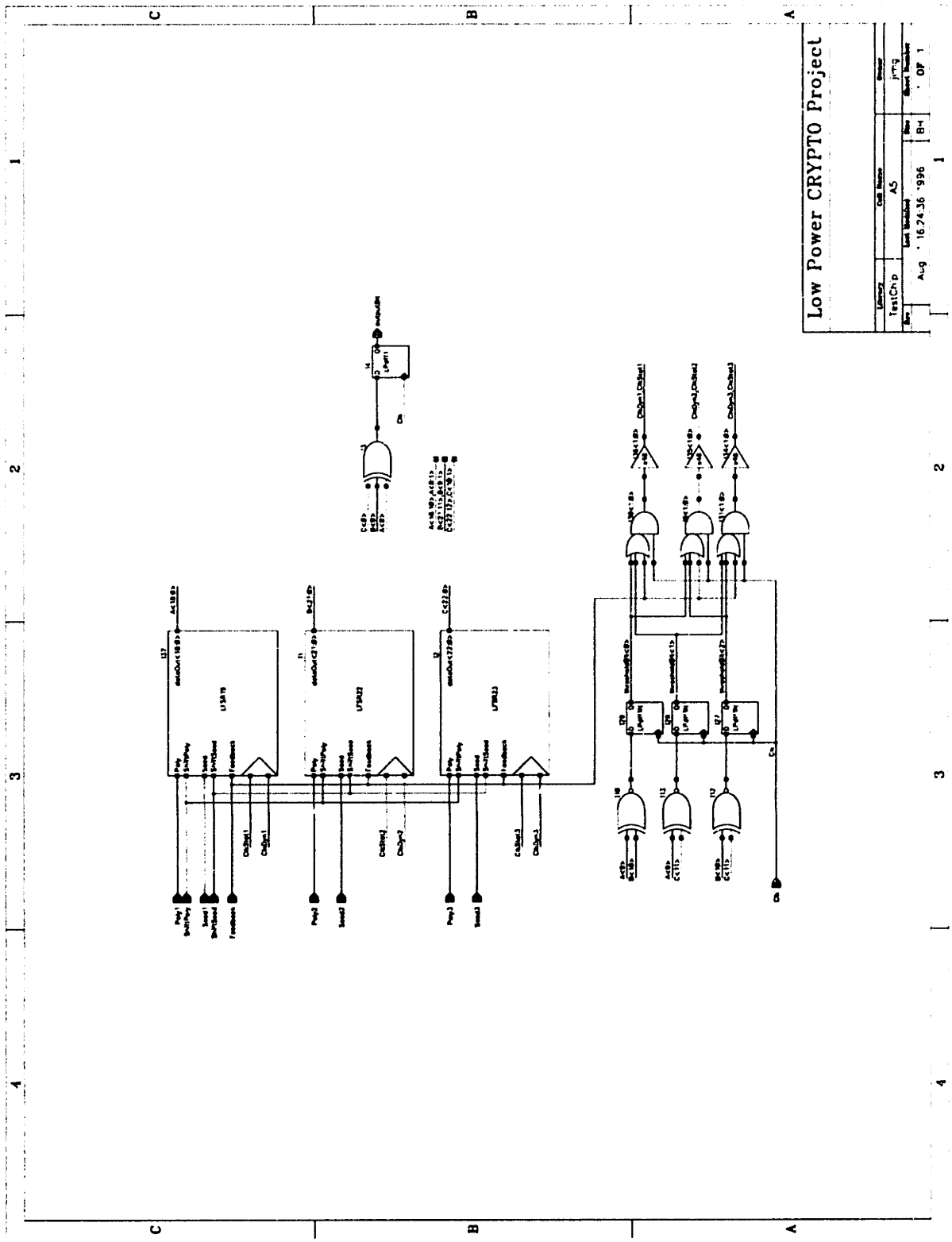



**Figure A-4:** Proposed GSM Cipher (A5) Schematic

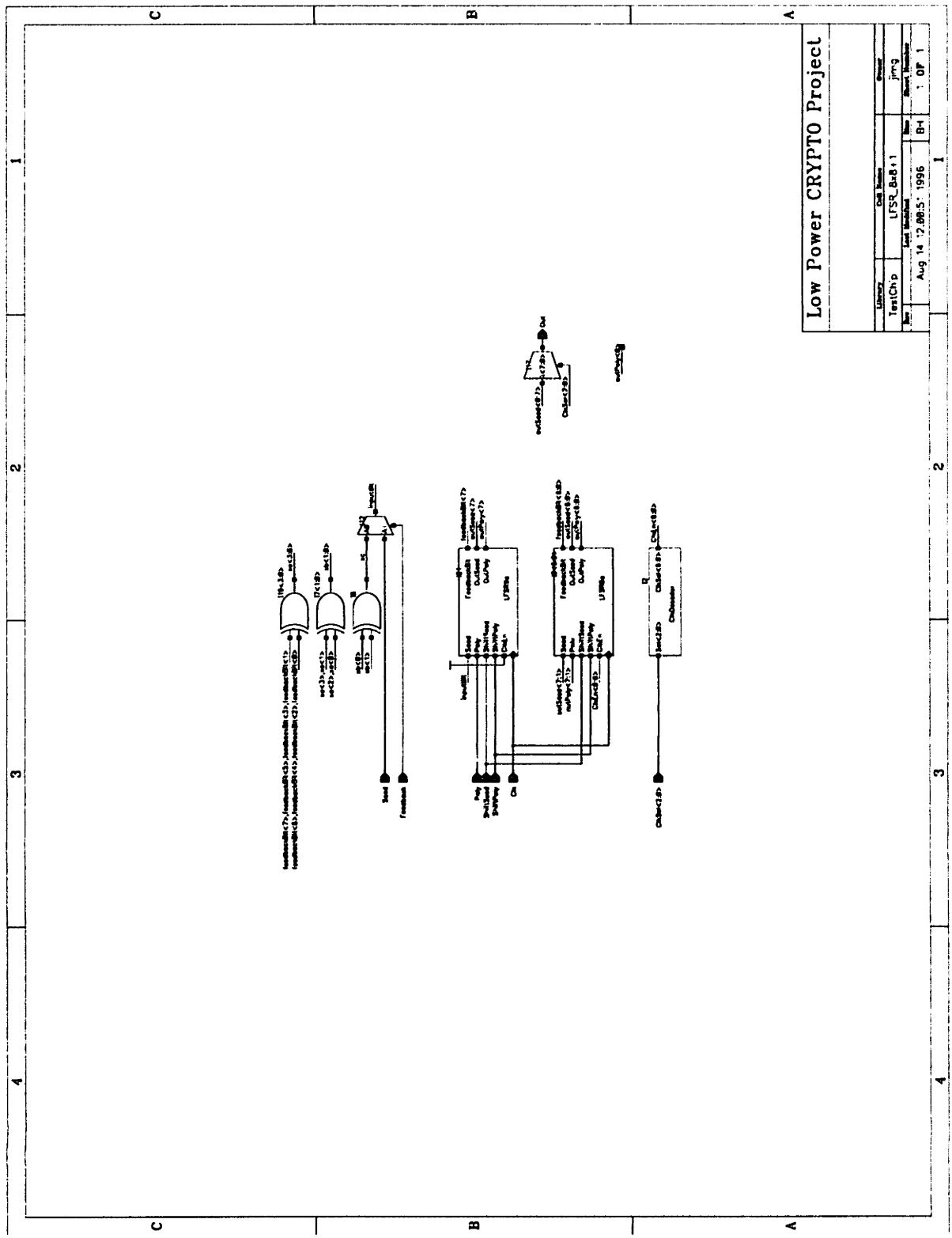



**Figure A-5:** 65-bit Variable Length Programmable LFSR Schematic

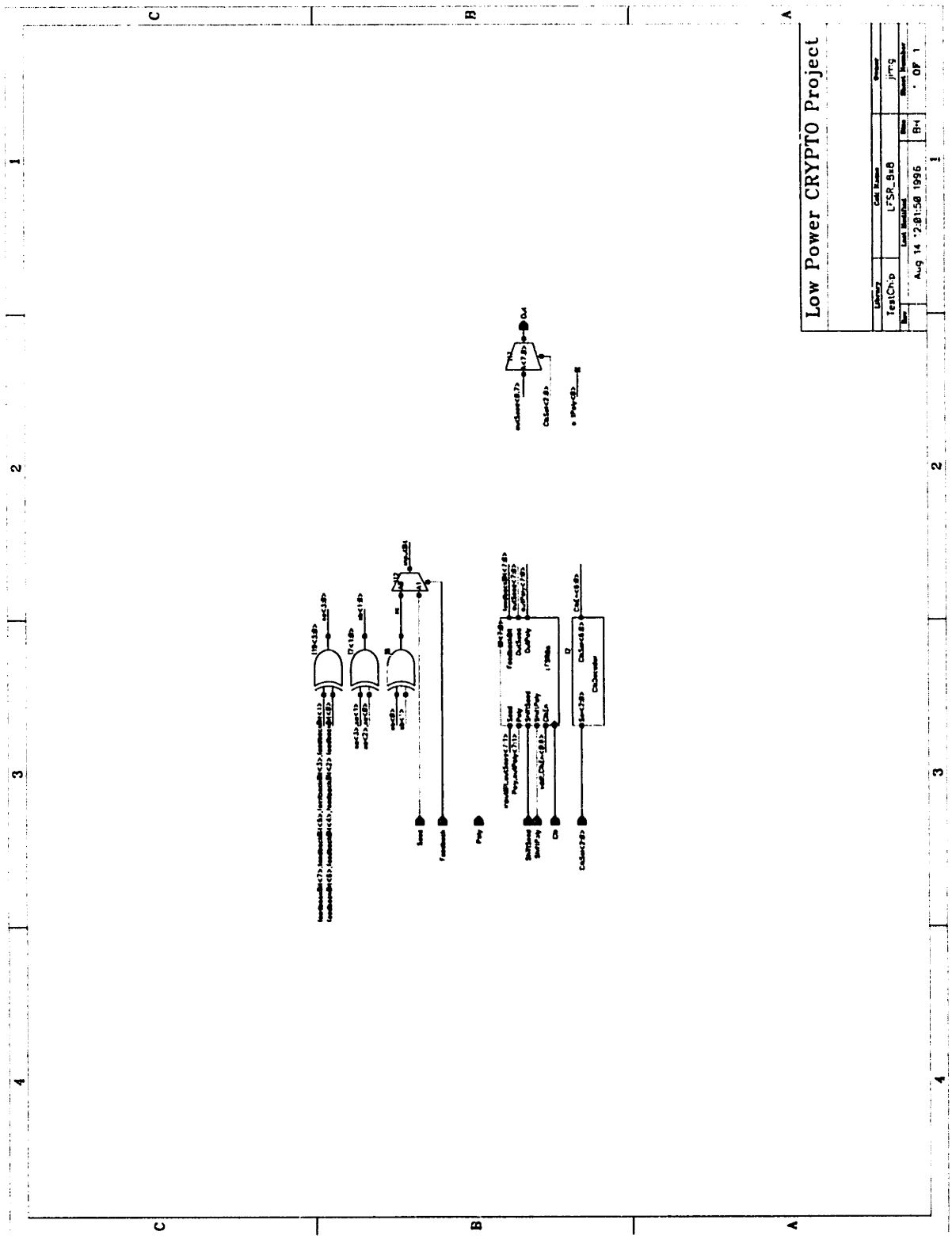



**Figure A-6:** 64-bit Variable Length Programmable LFSR Schematic


**Figure A-7:** 63-bit Variable Length Programmable LFSR Schematic

# Appendix B

# Modular Multiplier Schematics

This section provides the schematics for the major functional blocks within the Modular Multiplier Test Chip, down to the individual logic blocks of the bitslices and recoders. The internal schematics of the redundant adders, recoders and selectors are not provided as they are transistor-level schematics whose layout represents the floorplan of the physical layout for each cell. A list of provided schematics is given below along with a brief description of each.

- TakagiTop - top level design schematic
- 8bitController - the control logic of the modular multiplier
- SerialIO - the serial test interface
- CjSelector - the quotient estimation unit
- YRecoder - Y operand recoder
- MultDatapath - all of the bitslices and interconnections of the Bitslice Core
- BitsliceLow_layout - bitslice used for all but the most significant digits
- BitsliceHigh_layout - bitslice used for only the most significant digits

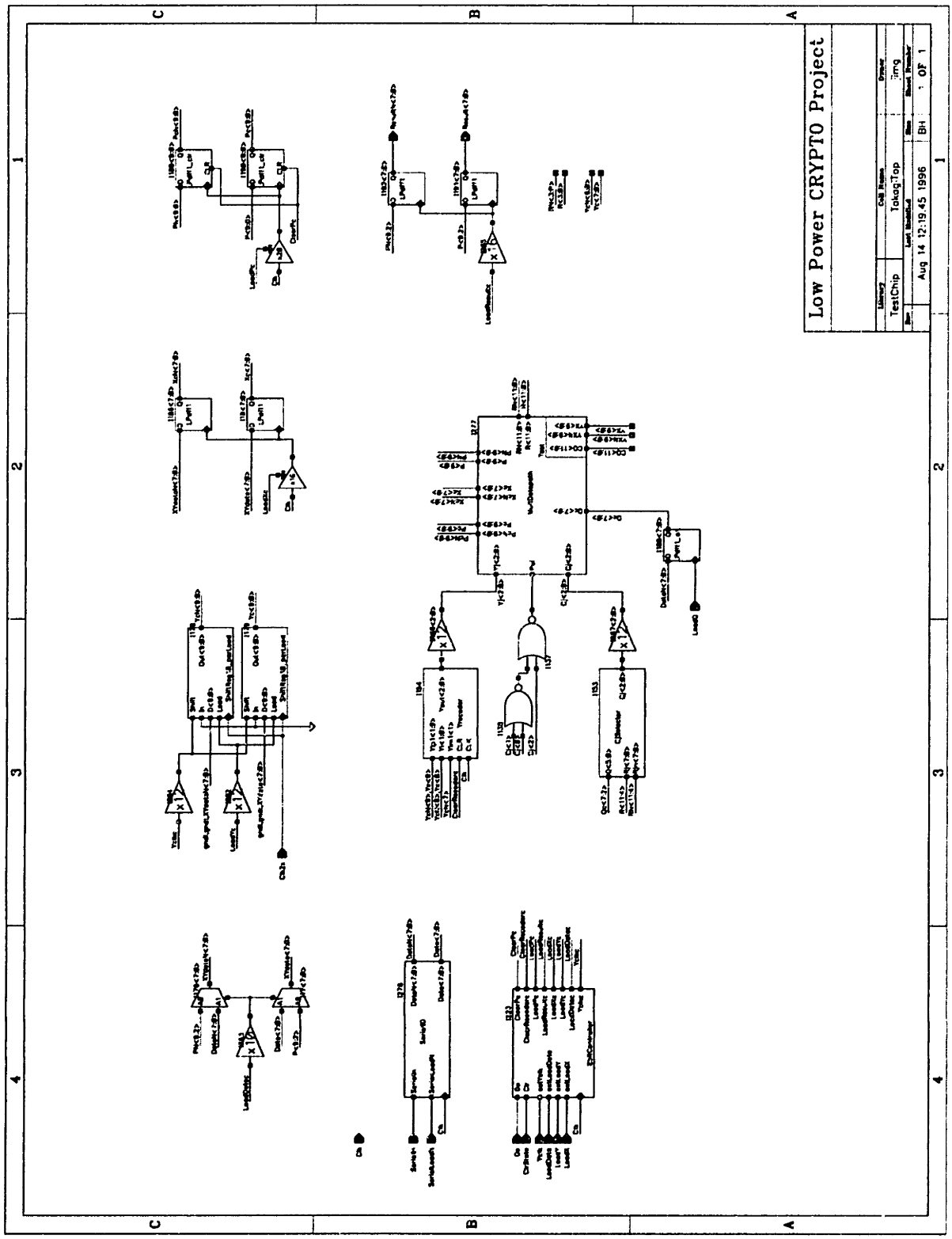



**Figure B-1:** Top Level Modular Multiplier Schematic

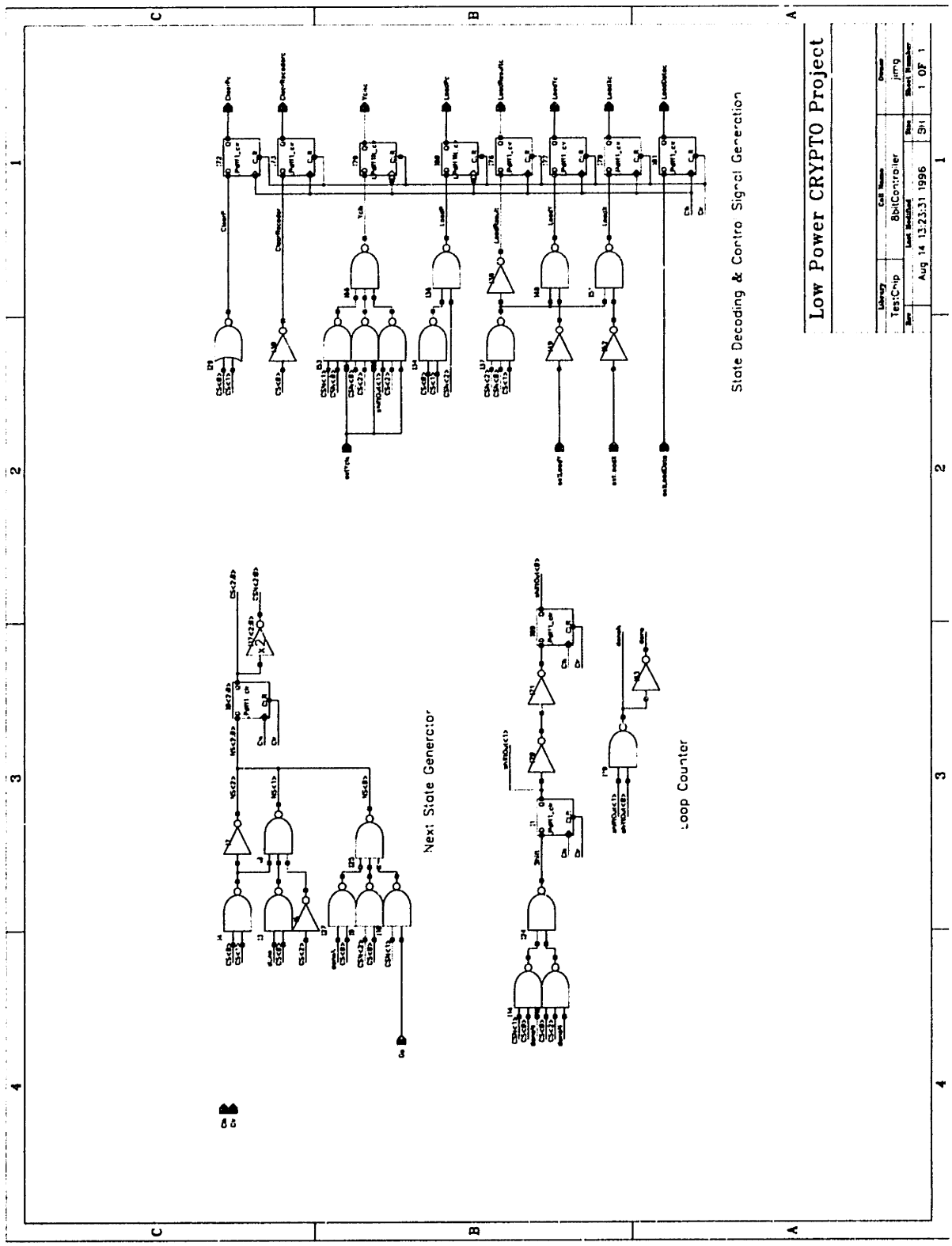



**Figure B-2:** Controller Logic Schematic

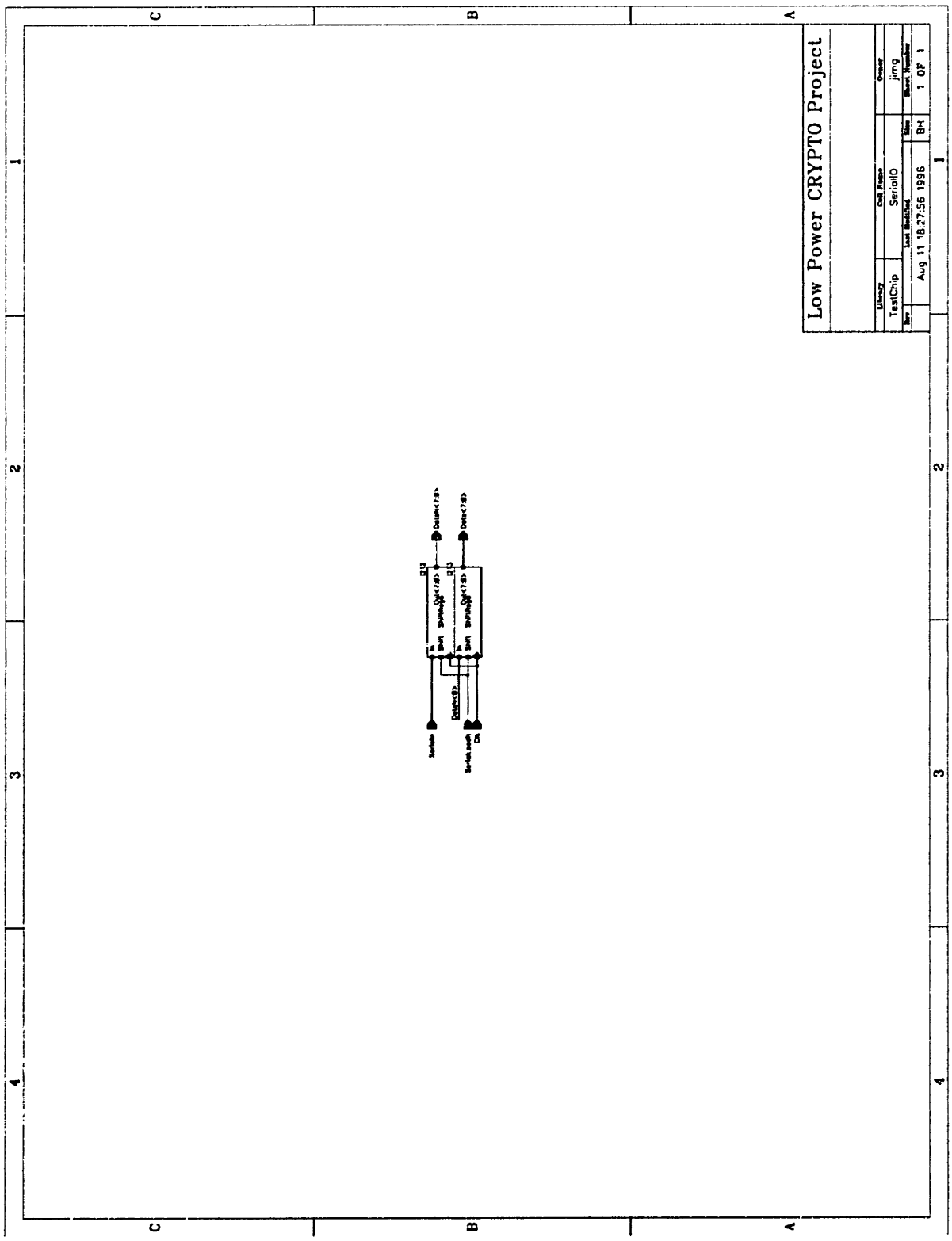



**Figure B-3:** Serial I/O Interface Schematic

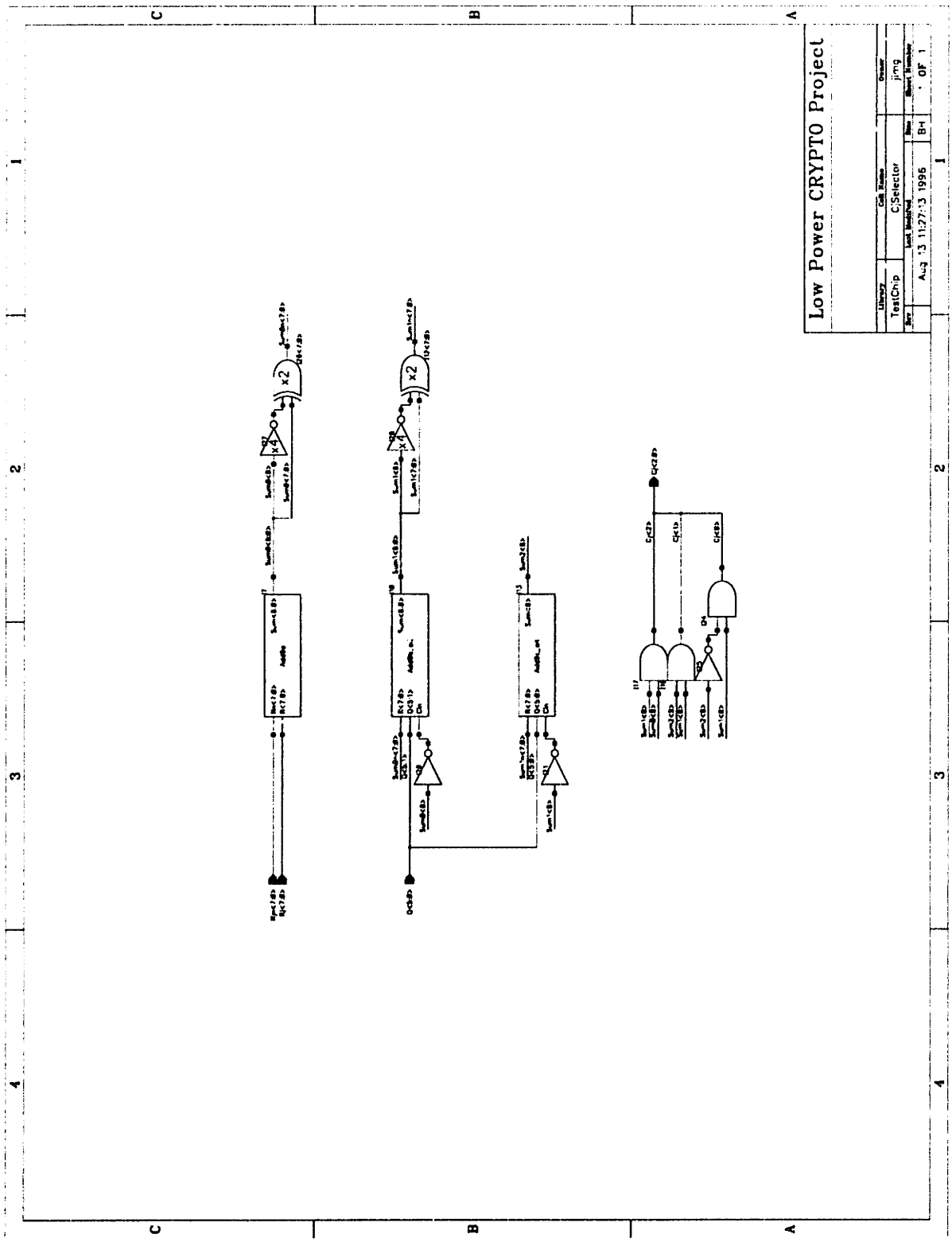



**Figure B-4: $C_j$ Selector Schematic**

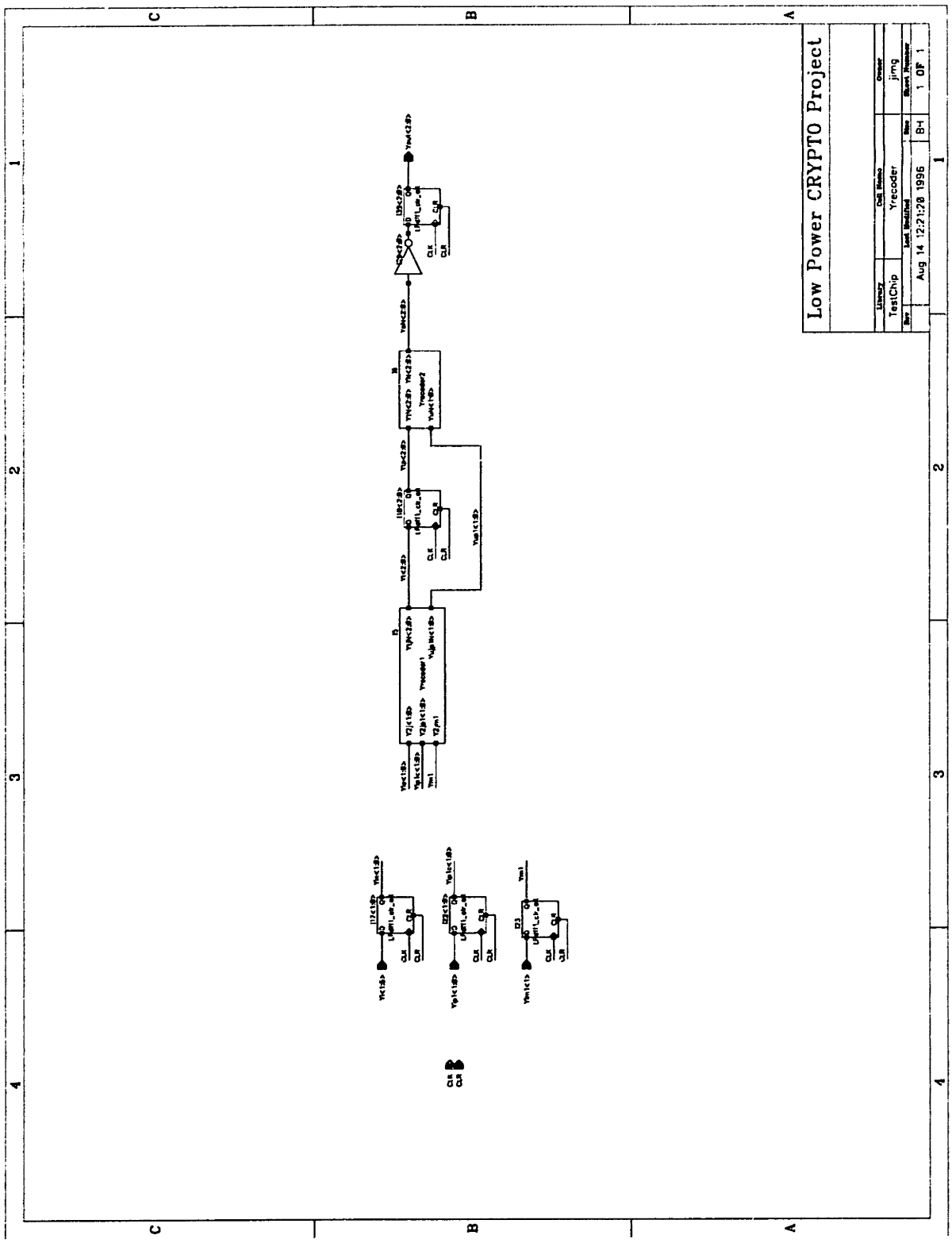



**Figure B-5:** $Y_j$ Recoder Schematic

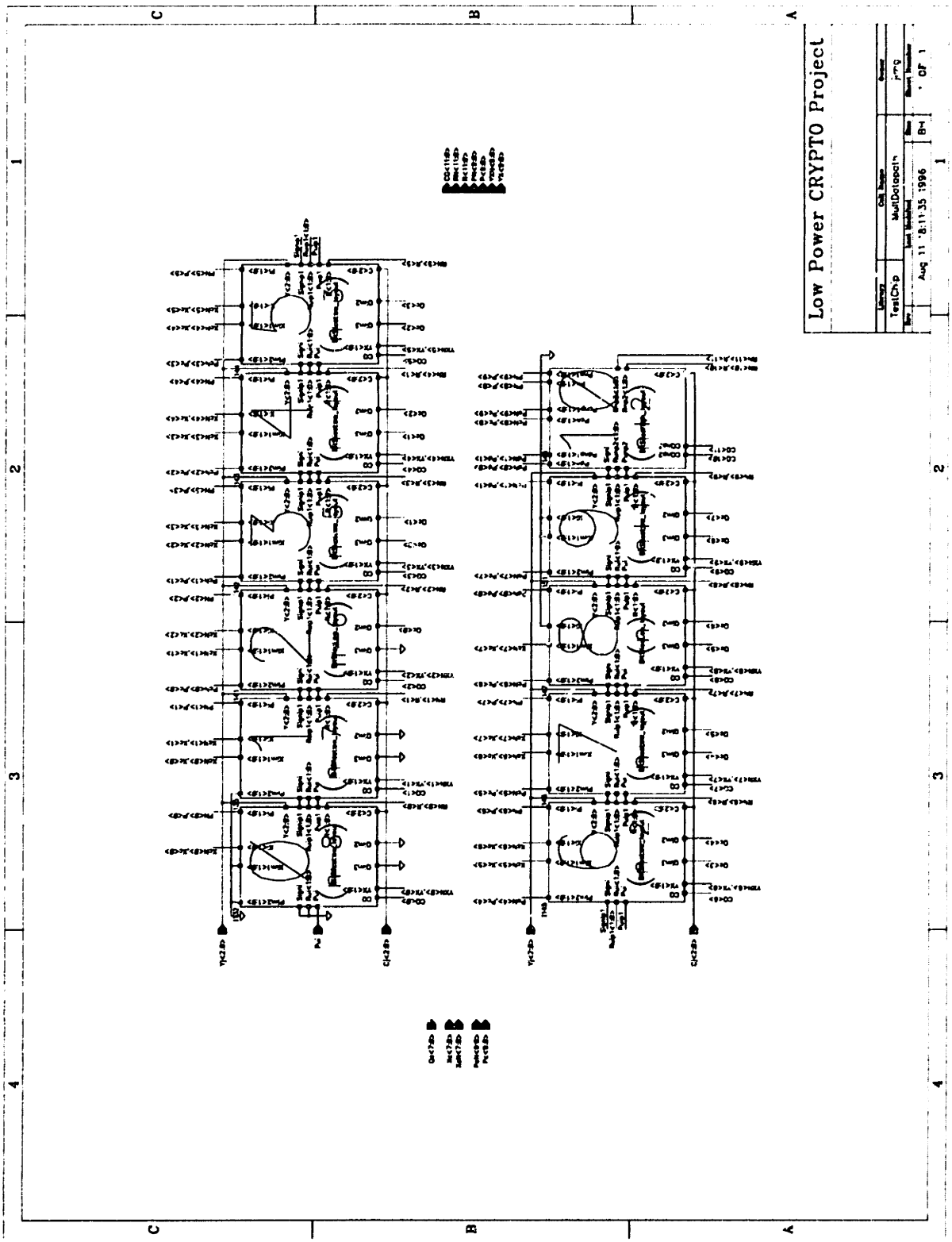



**Figure B-6:** Modular Multiplier Datapath Schematic

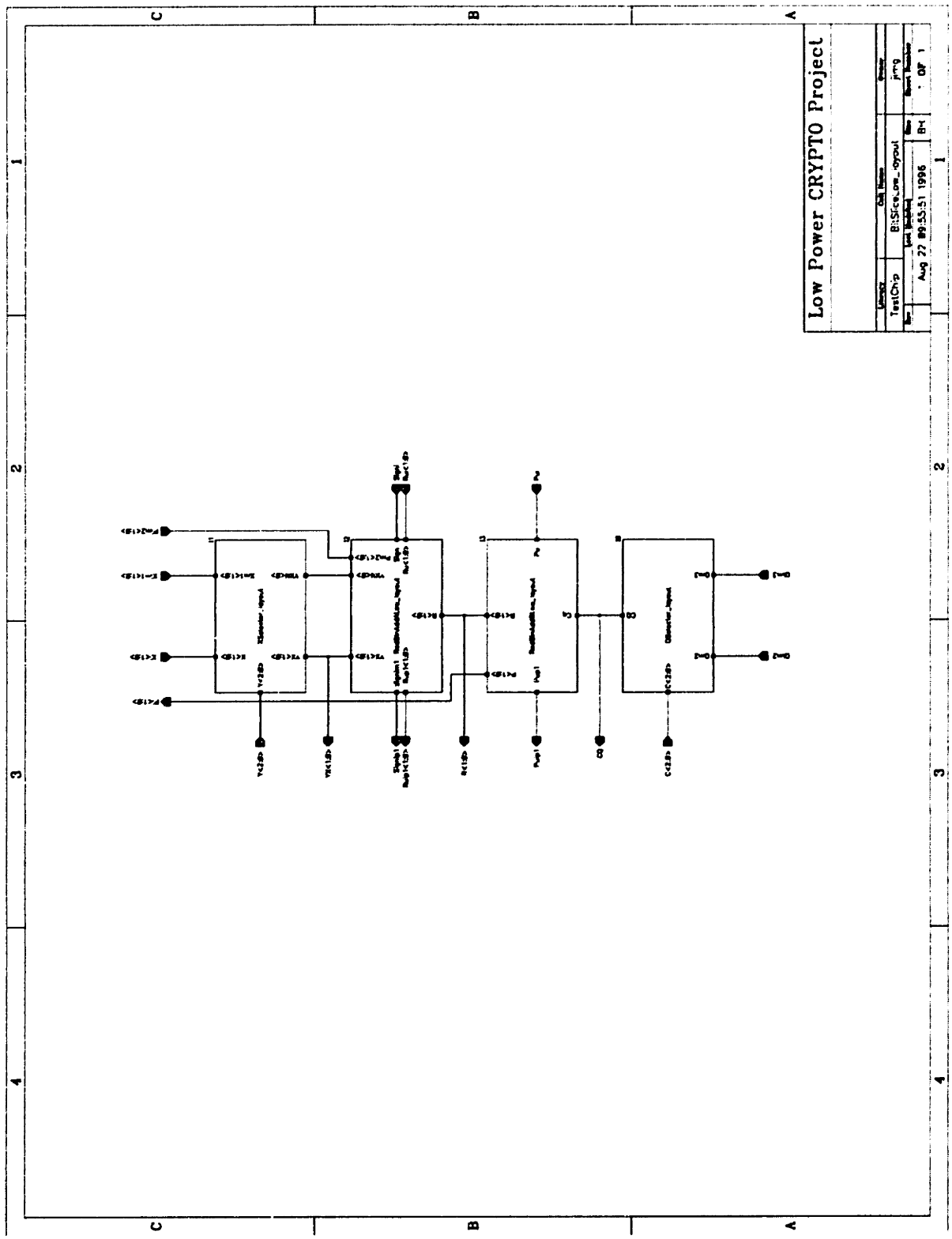



**Figure B-7:** Low Bitslice Schematic

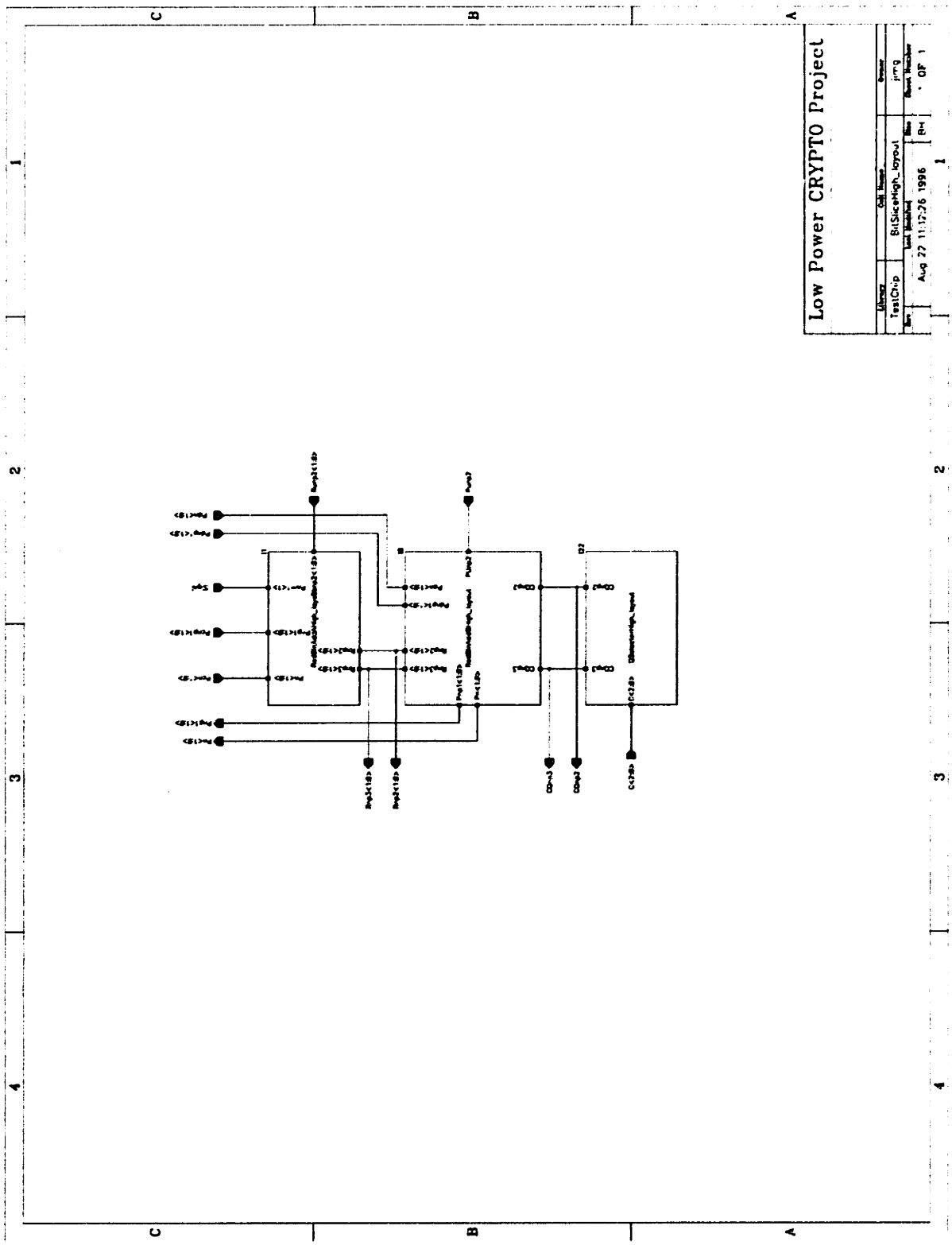



**Figure B-8:** High Bitslice Schematic