Hardware Performance Simulations of Round 2 Advanced Encryption Standard Algorithms

Bryan Weeks, Mark Bean, Tom Rozylowicz, Chris Ficke

National Security Agency

Abstract

The National Security Agency (NSA) is providing hardware simulation support and performance measurements to aid NIST in their selection of the AES algorithm. Although much of the Round 1 analysis focused on software, much more attention will be directed towards hardware implementation issues in the Round 2 analysis. As NIST has stated, a common set of assumptions will be essential in comparing the hardware efficiency of the finalists. This paper presents a technical overview of the methods and approaches used to analyze the Round 2 candidate algorithms (MARS, RC6, RIJNDAEL, SERPENT and TWOFISH) in 0.5um CMOS-based hardware. Both design procedures and architectures will be presented to provide an overview of each of the algorithms and the methods used. To cover a wide range of potential hardware applications, two distinct architectures will be targeted for comparison, specifically a medium speed, small area iterated version and a high speed, large area pipelined version. The standard design approach will consist of creating hardware models using VHDL and an underlying library of cryptographic components to completely describe each algorithm. Once generated, the model can be verified for correctness through simulation and comparison to test vectors, and synthesized to a common CMOS hardware library for performance analysis. Hardware performance data will be collected for a variety of design constraints for each of the algorithms to ensure a wide range of measured data. A summary report of the findings will be presented to demonstrate algorithm performance across a wide range of metrics, such as speed, area, and throughput. This report will provide a common baseline of information, which will enable NIST and the community to compare the hardware performance of the algorithms relative to one another.

1 Introduction

The National Security Agency (NSA) agreed to provide technical support to the National Institute of Standards and Technology (NIST) in the form of an analysis of the hardware performance of the Round 2 Advanced Encryption Standard (AES) algorithm submissions. This analysis consisted of the design, coding, simulation and synthesis of the five algorithms using the procedure outlined below. Throughout this evaluation, NSA has taken care to assure that best design practices were used and that all algorithms received equal treatment. No attempt was made to optimize any particular design, but care was taken to find the best configuration for each algorithm. Cross-validation measures during design and simulation were used to overcome the subjective effects of the design process and to ensure that all designs receive the same amount of attention. The results of this analysis should provide an accurate measure of the hardware performance of each algorithm relative to the others. Undoubtedly more optimized (and hence better performing) implementations of these algorithms can be designed, so the individual score of any particular algorithm is not very valuable outside the context of this environment. The point of this analysis is to provide a controlled setting in which a meaningful comparison can be made.

Based on a mathematical description of the Round 2 algorithms, and C code reference models when necessary for clarification, NSA designers fully described each of the algorithm submissions in a hardware modeling language. A review by a team of design engineers followed the initial design stage to reduce the effects of coding style on performance. Using commercially available analysis, simulation and synthesis tools, NSA design engineers have performed simulations to produce performance estimates based on each of the hardware models. In order to provide a wider perspective on the performance of the algorithms, two different architectures or applications were simulated for each algorithm: an iterative version to provide a medium speed operation at minimal area/transistor count, and a pipelined version to provide optimum speed operation, but at the cost of a larger area. This report is a summary of the performance of the Round 2 AES candidate algorithms, and will compare and contrast the results of the analysis.

2 Hardware Design Background

2.1 Design Guidelines

For this analysis effort, one of the main goals was to provide an unbiased comparison of the algorithms in hardware, specifically in Application Specific Integrated Circuits (ASICs). To that end, the overhead found in typical hardware implementations, such as a robust user-interface, was minimized to reduce the impact on the overall performance of the algorithms. The user-interface is the Input/Output (I/O) connections and logic needed to take the plaintext and key and present them to the algorithm, and take the output ciphertext and present it off the chip. All inputs and control signals were registered in a common interface in order to provide uniformity across all of the algorithms, with fixed setup and hold times identical for all algorithms. A wide variety of architectures could be used to implement a given algorithm. In order to restrict all possible choices and yet capture valuable data points, two fundamental architectures were chosen: iterative and pipelined. All algorithms were designed in each architecture style. There are several variations on these approaches, including multiple copies of an iterative implementation for parallel processing, a partially pipelined implementation, or a combination of these hybrids (multiple copies of a partially pipelined implementation). The approach chosen will depend on the needs of the system, but these variations will likely result in performance within the ranges given by the iterative and fully pipelined implementations. However, these optimizations were beyond the scope of this study.

2.1.1 Target Applications

2.1.2 Iterative Architecture

The iterated approach to implementing the algorithm focuses on providing a medium to low speed version of the algorithm, with efforts placed on limiting the physical size of the hardware. In this instance of the algorithm, one step is performed per clock period, with the output of the previous step being used as the input to the next step. Data is only placed on the output after the required number of algorithm rounds has been completed.

2.1.3 Pipelined Architecture

The pipelined approach to implementing an algorithm centers on providing the highest throughput to the design, sacrificing area to obtain the level of performance needed. In the case of pipelining, all of the steps in computing the algorithm are cascaded into a single design, with each stage feeding the next stage. The latency remains the same as in the iterated case, but the throughput is increased significantly as new data is placed on the output on every clock cycle. Pipelining has been shown to be an effective method of dramatically increasing the throughput capabilities of a given algorithm. However, it comes at the expense of limiting the number of cryptographic modes that can be supported at the maximum throughput rate. For example, since the latency of an encryption cycle remains the same as an iterative case, there is no throughput advantage when using feedback modes such as Cipher Block Chaining (CBC). High performance applications, such as high speed network encryption, will require the increase in throughput, and as a result, often focus on a non-feedback mode of operation such as counter mode to obtain performance.[1]

2.2 Parameter Description

There are many design parameters that can be reported for each design implementation. Some parameters will have much more significance in a given application or environment than others. This evaluation reports on these parameters as a method of comparison among the five algorithms, and does not claim that any single parameter has been fully optimized. The following is a description of the parameters being reported. Some have a direct impact or relation to performance metrics (e.g. throughput) and some are simply a function of the algorithm itself (e.g., I/O requirements). Algorithm performance in each of the evaluation categories will be documented for each algorithm submission.

2.2.1 Area

As an estimate based on an available MOSIS library, the results of the synthesis area reporting will consist of pre-layout area estimates of the algorithm. Although potentially different from a post-layout estimate, the area reported by Synopsys (version 1999.10) will provide a relative comparison of each of the algorithm submissions. Generally, the two varieties of architectures-– iterative and pipeline -– will be on the extremes of area with the iterative being the smallest, and the pipelined being the largest.

2.2.2 Throughput

In most cases, throughput is directly proportional to area; as area decreases, throughput decreases. As with area, the iterative and pipelined architectures will report the extremes of throughput. Iterative architectures will have much lower throughput rates since there is a minimum amount of hardware, and it is re-used on multiple clock cycles of execution. Thus, the throughput is limited by the amount of hardware reuse. More specifically, it is limited by the number of rounds in a codebook algorithm. On the other hand, a pipelined architecture dedicates hardware for performing all calculations in any given clock cycle. This maximizes throughput by allowing data to be written and read from the device on every clock. In this case, throughput is a function of the worst-case delay in any one given stage of the algorithm. Throughput will be reported for both iterative and pipelined architectures.

2.2.3 Transistor Count

Transistor count is a more specific measure than area and is often more useful. While transistor count is somewhat dependent on the design library being used, it is a useful method of comparing the algorithms since they were compiled using the same library. In addition, the transistor count will be a more useful figure than area when estimating programmable logic implementations since these devices typically report the number of useable gates (which is also directly related to transistor count). Based on the synthesized netlist (from Synopsys), an additional report describing the number of transistors required to implement the algorithm will be provided.

2.2.4 Input/Outputs (I/O) Required

With the goal of consistency among algorithms, the I/O was fixed identically for all algorithms. However, since this parameter is highly useful to hardware designs, it will still be reported.

2.2.5 Key Setup Time

The key setup time refers to the amount of time required before subkey expansion is ready to execute. Some algorithms use the user-supplied key directly in the subkey expansion thereby reducing the key setup time to zero. Others require some pre-calculation or translation of the key prior to subkey expansion steps. Key setup times will be examined to assess the overhead of each algorithm in establishing a usable key.

2.2.6 Algorithm Setup Time

Similar to key setup time, the algorithm setup time reports the minimum amount of time before an algorithm is ready to process data. Time to create look-up tables, etc. will fall in this category. None of the evaluated algorithms contained an algorithm setup time greater than zero.

2.2.7 Time to Encrypt One Block

This paramter will address minimum latency times for each of the algorithm submissions. The time to encrypt one block, measured in nanoseconds, is a function of two parameters: the worst-case path delay between any two registers, and the number of rounds in the algorithm.

2.2.8 Time to Decrypt One Block

As above, this parameter will address minimum latency times for each of the algorithm submissions. Decryption does not always require identical processing as encryption. Therefore, the time required to decrypt one block is reported.

2.2.9 Time to Switch Keys

Originally, this parameter was included as a measure to encompass both key setup time and algorithm setup time overhead. However, since none of the evaluated algorithms contained an algorithm setup time, this parameter is identical to key setup time. Therefore, it will not be reported further in this document.

3 Methodology

3.1 Standard Design Flow

The design process followed a common methodology used by ASIC designers. The process started with the documentation supplied by the algorithm authors and was completed with a gate-level schematic, which included the performance metrics data. A complete ASIC development would require physical layout and fabrication. These steps were beyond the scope of this effort. However, the performance metrics data obtained here closely matches that which would be found from actual fabrication and testing. Previous efforts using these tools have correlated estimated performance from the schematic to the actual testing. Figure 1 shows the steps in the design flow.

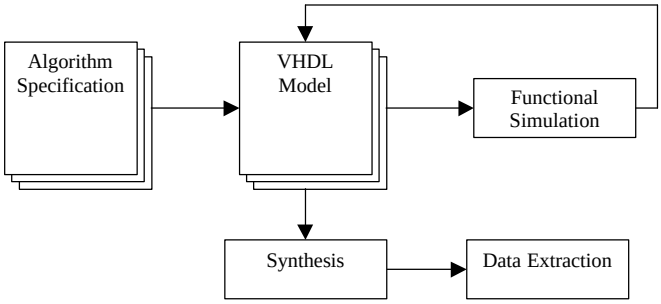

Figure 1 Standard Design Flow

3.1.1 VHDL code generation

VHSIC Hardware Description Language (VHDL)

VHDL modeling is analogous to programming simulations in C code and follows much of the same syntax. However, unlike a behavioral description of the algorithm, VHDL (IEEE 1076) specifies how the algorithm will be implemented in hardware. Using this hardware language, NSA designers fully described the hardware necessary to implement each of the algorithm submissions. Performance metrics, such as speed, area, etc. (see below) can be estimated from the hardware description using available analysis and computer aided design (CAD) tools.
There are different styles in which to code VHDL models, offering various levels of abstraction. For this evaluation, the designers used the register transfer logic (RTL) coding style. For this style, the placement of registers and corresponding logic between registers is chosen by the designer and is determined at the VHDL code level. There are many different methods for identifying an optimized placement of registers. Ideally, there would be an equal amount of logic delay between registers for all stages of the design. However, in order to simplify the design cycle and to be consistent among all algorithms, the designers chose a common placement of registers, even if this placement is not fully optimized. Specifically, the output of each "round" (as defined by the algorithm authors) is registered for both a key schedule round and an algorithm round.

3.1.2 Simulation and Verification

NSA followed the design phase with a functional VHDL simulation of the designs using the Synopsys VHDL System Simulator (VSS) to verify the correct operation of the algorithm. The test vectors submitted to NIST for each algorithm were applied to assure that the design was working as intended. Specifically, the Variable Key and Variable Text tests were performed for each algorithm implementation and mode (e.g., iterative encrypt, pipelined decrypt, etc.). The modeled algorithm output was also compared with the C code model supplied to provide an added assurance that the simulation was operating as expected.

3.1.3 Code review

NSA had one or more engineers design the VHDL for each algorithm submitted. Initial hardware designs were straightforward implementations of the core algorithm. Following completion of each initial design, an informal group of engineers met to review and provide feedback for the design. Improvements and alternatives to the initial design were examined to determine potential benefits from differing architecture approaches (area compression, pipelining, etc.). Variants of the design that improve the performance of the algorithm were then programmed for comparison.

3.1.4 Synthesis

Gate-level synthesis of the algorithm utilized the Synopsys Design Compiler to produce a functionally equivalent schematic in hardware. A MOSIS-specific technology library was used to generate a gate-level schematic of the design and provide more accurate area and timing estimates, as if the design were to be implemented in an integrated circuit (IC). The MOSIS library is based on a publicly available fabrication facility's model of a specific CMOS process (0.5um), thus giving real performance metrics for an available ASIC line. The VHDL model can be re-targeted to any supported hardware or field programmable gate array (FPGA) design libraries.

The synthesis process can generate a wide range of implementations depending on the constraints provided to the synthesis tool. For example, one implementation may minimize area while another may minimize delay time. In hardware synthesis, the two fundamental parameters are time and area. These parameters are directly related. As delay time decreases, area increases. Timing and area curves that further illustrate this point are shown in subsequent sections. The constraints provided for each algorithm synthesis routine were maintained consistently. Therefore, differences among algorithm synthesis results will be a function of the logic required (algorithm specific) and the synthesis tool's ability to meet the given constraints.

3.1.5 Documentation

In addition to a summary report containing performance data, both design notebooks and VHDL documentation will be provided to NIST for evaluation. The design notebooks will contain reporting information for all of the hardware data that was collected, with all algorithms, designs, and architectures represented. The VHDL models and their testbenches (for simulation verification) will also be included.

3.2 Synthesis Analysis

3.2.1 Function Characterization

Although the hardware design of the algorithms followed a top-down approach, the synthesis portion of the analysis proceeded from the bottom of the design up through the hierarchy. In order to obtain an accurate picture of the performance of the sub-blocks and functions in this type of analysis, a sweep was performed on each of the functional blocks to graphically depict performance versus design constraints. Specifically, the timing constraints, such as output delay and clock frequency, of each of the blocks were varied to observe the performance output of the block. The results of the sweep make up the characterization for that particular block. All subsequent blocks of the hierarchy will be analyzed using these methods.

3.2.2 Cryptographic Library

With characterization curves for each of the sub-blocks complete, five speed grade implementations were selected to cover the performance range of the block. A variety of key performance points were selected to reflect requirements for both high speed and small area. Figure 2 shows typical timing and area curves following a sweep of maximum delay time constraints. These curves allow design engineers to select specific implementations of a given function. Specifically, five implementations, or speed grades, were chosen for each function. In this example, the five selected implementations are noted in the figure, and they represent one minimum time delay, one minimum area, and three other points that have desired characteristics such as large area savings for a small increase in delay time.

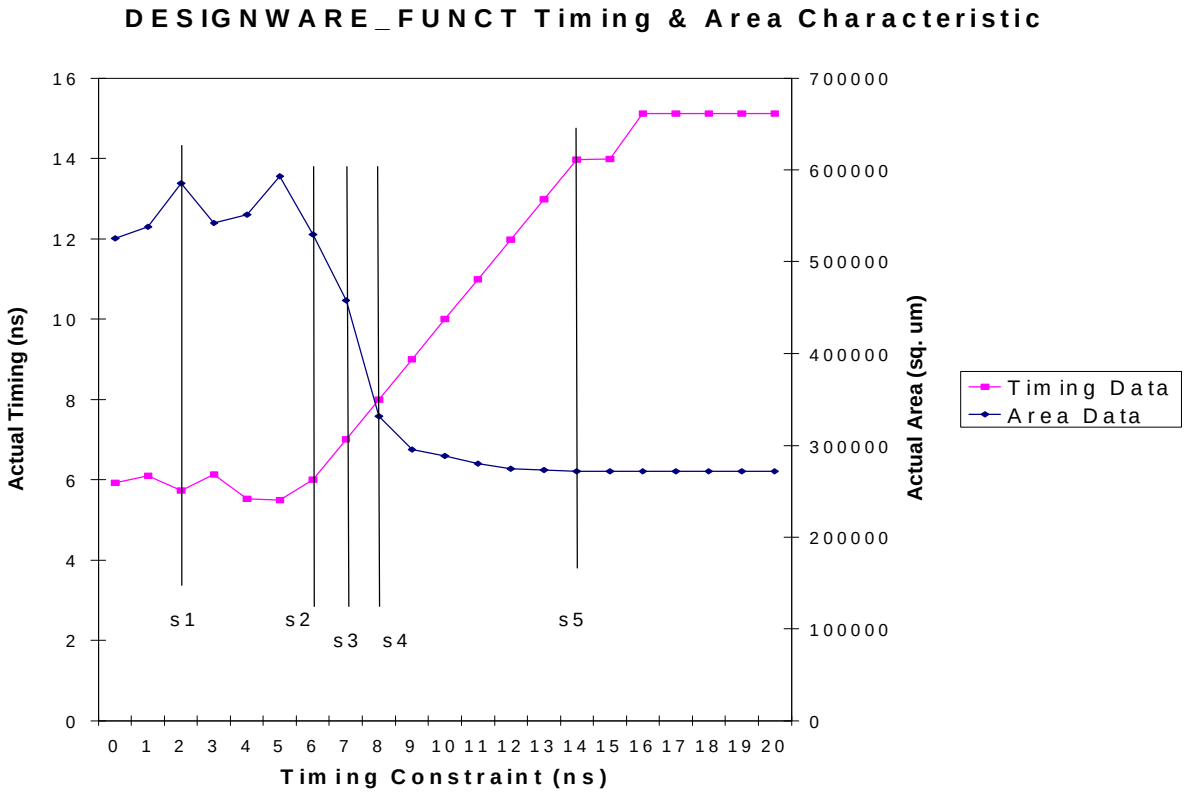

Figure 2 Sample Function Sweep

The five implementations were selected only for the functions of each of the algorithms, and then assembled into a cryptographic library. Each library contained implementations of all the functions required to build the given algorithms.

3.2.3 Block Level Characterization

Continuing with the bottom up approach, the higher level blocks underwent the same performance sweep as described for the function level. The design constraints were varied across the entire range of the block to fully describe the performance curve of the block. At each iteration of the synthesis process, components (e.g., functions) were selected from the cryptographic library based on the required speed and performance. Performance curves at the block level encompassed components of several different speed grades depending on design constraints. At the top level, the characterization curve reflected the performance of the entire design across a wide range of design constraints.

4 General Architecture Approach

4.1 Top Level Architecture

The design of each algorithm started with a common top level architecture that is well suited for virtually any codebook algorithm. The generalized top level architecture consists of an Interface, an Algorithm block, a Key Schedule block and a Controller. Figure 3 shows a block diagram of the architecture.

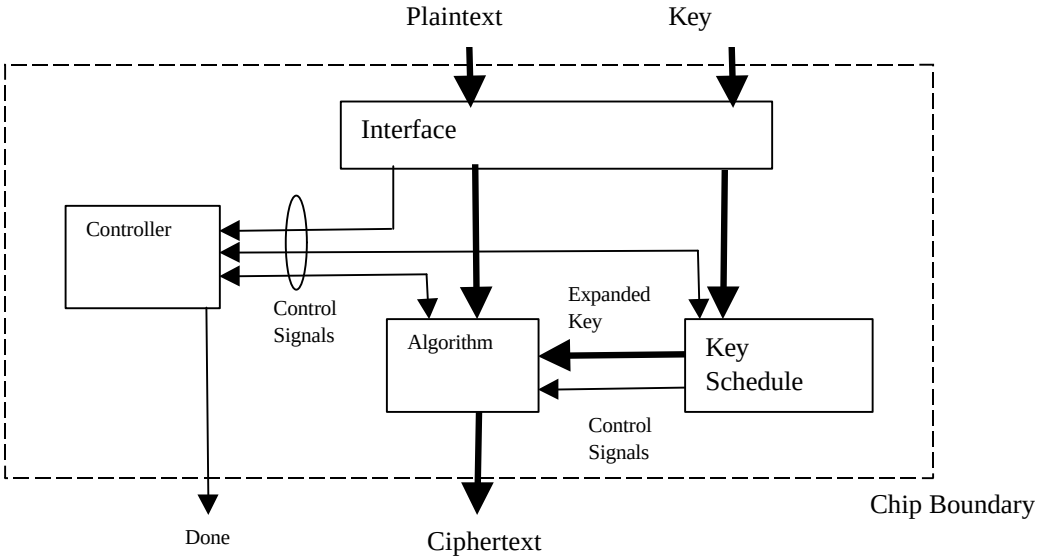

Figure 3 Top Level Architecture

The Interface serves to register all data inputs. This is consistent with the hardware design methodology of placing registers at the chip boundary, thus minimizing strict setup-and-hold timing requirements. In some cases, the interface also provides minimum functionality, such as padding keys when appropriate. The Key Schedule performs the generation of subkeys to be used in by the Algorithm. This includes any required key setup as well as the expansion itself. The Algorithm performs the actual encryption or decryption of data provided from the Interface using the subkeys from the Key Schedule. For iterative implementations, the Algorithm and Key Schedule implement a single round with internal feedback datapaths; the pipelined implementations expand these sections to include as many implementations of a round as required by the particular algorithm. Finally, the Controller provides any necessary control signals for maintaining proper synchronization among the various blocks.

5 Algorithm Evaluation

A description of how each algorithm was architected is given for both the pipelined and iterative cases. Any nuances of how the rounds were simulated and the key schedule implemented are also given along with specific examples of approaches to reduce redundancy or streamline the design. Following the architecture for each of the algorithms, each section will provide a summary of the results of the hardware analysis for the individual algorithm.

A table of performance parameters is then provided for two different key sizes -- the 128 bit and a hybrid that combines all three key sizes in one key schedule that can be controlled for any particular key size. In some cases, the combined three-in-one key schedule must make compromises to achieve the greater degree of flexibility. Each of the performance parameters is described in more detail in Sections 6-7, along with comparisons across the five algorithms.

5.1 MARS

5.1.1 Architecture

The MARS algorithm requires several different types of rounds[2]. Specifically, there are unkeyed forward mixing, keyed forward transformation, keyed backwards transformation and unkeyed backwards mixing rounds, as well as pre-addition and post-subtraction. The mixture of keyed and unkeyed rounds resulted in the requirement for complex control and data flow operations between the Key Schedule and Algorithm blocks. Specifically, a complex control situation results from the fact that subkeys are required immediately for the pre-addition stage, whereas the next subkeys are not required until the eight unkeyed forward mixing rounds are completed. This architecture presented some unique timing and data synchronization issues.

5.1.1.1 Pipelined Key Schedule

As with all pipelined implementations, the subkey from each round must be registered. However, for MARS, the subkeys are not utilized on consecutive clock cycles, making additional pipelined storage necessary. The updated key schedule of MARS following AES Round 1 allowed for separating the 40 subkeys into groups of 10. The VHDL model takes advantage of this operation by adding pipelined storage for groups of 10 subkeys, only as long as necessary, rather than creating pipelined storage for all 40 subkeys. This reduced the total number of registers required. Additionally, the pipelined registers are controlled by a latch signal rather than updating on every clock. Again, this reduced the number of registers by removing redundancy.

5.1.1.2 Pipelined Algorithm

Relative to the intricacies of the key schedule, the pipelined algorithm implementation is straightforward. It consists of six different types of rounds, each one with its own registered output: one key addition, eight unkeyed forward mixings, eight keyed forward transformations, eight keyed backwards transformations, eight unkeyed backwards mixings and one key subtraction. This makes a total of 34 rounds to complete the algorithm.

5.1.1.3 Iterative Key Schedule

The MARS algorithm key schedule generates 10 subkeys at a time. Therefore, the traditional iterative methodology of a single round implementation for generating a single subkey (or set of subkeys as required by a single algorithm round) did not apply. Instead, a single round implementation per 10 subkeys was generated resulting in a key expansion round iterated four times for one encryption cycle. This presents some additional logic overhead for iterative applications in that a “round” generates 10 subkeys simultaneously rather than the exact amount needed by the algorithm at a given stage. (In the case of MARS, two 32-bit subkeys are required per keyed round in the cryptographic core.) In addition to the subkey expansion overhead, there is a storage overhead for the remaining subkeys. Also, decryption requires a full expansion of subkeys prior to beginning data processing. Therefore, the full set of 40 subkeys is stored in registers.

5.1.1.4 Iterative Algorithm

The iterative algorithm is consistent with the pipelined algorithm in its relative simplicity when compared to the key schedule. There is a single register for all rounds. The input to the register depends on the round number. For example, the input for the first round of encryption is the key addition round result; for the second round it is the unkeyed forward mixing round result and so on.

Subkeys are presented to the algorithm block as an array of all 40 subkeys. This differs from other iterative algorithm implementations that present only one subkey at a time. The rationale for this design was to eliminate duplicate logic in both the Key Schedule block and Algorithm block. Due to the timing gaps in the application of subkeys and the fact that all 40 subkeys are generated prior to decryption processing, it was considered advantageous to allow a 40 element bus to connect the two blocks.

5.1.1.5 Effects of Single Key Sizes

For MARS, the effects of implementing single key sizes is limited to the key expansion portion only. Both the algorithm and number of rounds will remain constant across key size. The effect of using the single key size was

marginal because the core function that changes was initializing T. This was relatively minor compared the much larger STIR/SBOX function which was independent of the key length.

5.1.2 MARS 3-in-1 Results

Parameter	Iterative 3in1		Pipelined 3in1	
	Min.	Max.	Min.	Max.
Area (um2)	52,717,560	127,432,766	670,181,619	1,333,099,627
Transistor Count	734,751	1,950,277	9,340,818	20,667,308
Input/Outputs Required	520	520	520	520
Throughput (Mbps)	29.06	56.71	1123.89	2189.16
Key Setup Time Encrypt (ns)	9553	27470	3718	10206.23
Key Setup Time Decrypt (ns)	9553	27470	3718	10206.23
Algorithm Setup Time (ns)	0	0	0	0
Time to Encrypt One Block (ns)	2256.92	8600.1928	1987.98	3872.26
Time to Decrypt One Block(ns)	2256.92	8600.1928	1987.98	3872.26

Table 1 MARS Summary

5.1.3 MARS 128 Bit Key Results

Parameter	Iterative 128-Bit		Pipelined 128-Bit	
	Min.	Max.	Min.	Max.
Area (um2)	51,986,292	126,827,662	669,735,987	1,332,658,283
Transistor Count	724,403	1,941,371	9,334,605	20,661,116
Input/Outputs Required	520	520	520	520
Throughput (Mbps)	29.06	56.71	1123.89	2189.16
Key Setup Time Encrypt (ns)	9553	27429	3712	10199
Key Setup Time Decrypt (ns)	9553	27429	3712	10199
Algorithm Setup Time (ns)	0	0	0	0
Time to Encrypt One Block (ns)	1987.98	3872.26	1987.98	3872.26
Time to Decrypt One Block(ns)	1987.98	3872.26	1987.98	3872.26

Table 2 - MARS 128 Bit Key Summary

5.2 RC6

5.2.1 Architecture

The following provides a high level description of the major blocks in the RC6 algorithm. Details of the components, sweeps, and their implementations can be found in the design workbook.

5.2.1.1 Pipelined Key Schedule

The RC6 key schedule is pipelined using a slightly different method than the other algorithms. Since a significant number of computations for the key schedule are required before any expanded keys are generated, the architecture takes advantage of the run-up by performing the expansion at the start of the pipeline. Only a single copy of the expansion hardware is required, but additional registering is needed to maintain the keys on a time dependent basis, discarding keys from previous stages (i.e., the keys have already been used). Keys are then passed from register to register to follow the data in the pipeline.

5.2.1.2 Pipelined Algorithm

The pipelined implementation "unrolls" the stages of the algorithm into a pipeline, following the algorithm description for function ordering and naming conventions. Combination functions are used to perform cases where distinct operations need to be performed in encrypt and decrypt. For example, the pre-add will contain both addition and subtraction to accommodate both cases. A similar condition exists in the algorithm round function, with slightly different functions needed for encrypt and decrypt. However, synthesis optimization can take advantage of common operations, such as the multiply, to reduce the total number of operators needed.

5.2.1.3 Iterative Key Schedule

The iterative key schedule is designed to perform a single round of expansion per clock. Expanded keys are fed to the algorithm block after the controller initiates a start signal. However, the key setup has been designed to compute single or multiple steps of the run-up in a single clock, depending on the performance needed by the rest of the system. A load cryptovariable (i.e., load key) signal from the controller will initiate the key setup. Once complete, the expansion can be started.

5.2.1.4 Iterative Algorithm

The RC6 iterative algorithm closely reflects the pipelined version. The same round instance is called repeatedly to process input data. The encrypt and decrypt are symmetrical with respect to operations performed in a similar manner (e.g., pre-add, round, post-add), so the same block can be called without additional overhead.

5.2.1.5 Effects of Single Key Sizes

The effects of single key sizes on RC6 were confined mostly to the key expansion portions of the algorithm. The core algorithm section was not greatly affected by a difference in key size, and the only noticeable changes occurred in the top-level algorithm design where different I/O requirements were used. The key expansion portions, however, showed the greatest changes in performance and area as a result of implementing different key sizes. In fact, every single layer in the key schedule design was affected by using a single key implementation. For the most part, these differences were a result of additional muxing and associated logic in the 3 in 1 case. However, there also exists a slightly greater memory requirement for storing larger keys along with associated look-up tables used to compute the array of sub-keys. These differences in area and performance, although minor at the lower levels, were greatly enhanced by the large number of iterations necessary to create the expanded keys.

5.2.2 RC6 3-in-1 Results

Parameter	Iterative 3in1		Pipelined 3in1	
	Min.	Max.	Min.	Max.
Area (um2)	16,027,545	21,660,006	318,931,799	592,088,503
Transistor Count	248,686	430,436	7,304,500	16,055,292
Input/Outputs Required	520	520	520	520
Throughput (Mbps)	54.94	103.98	1192.47	2170.96
Key Setup Time Encrypt (ns)	8,139.00	15,348.96	3,659.51	6,922.53
Key Setup Time Decrypt (ns)	8,139.00	15,348.96	3,659.51	6,922.53
Algorithm Setup Time (ns)	0	0	0	0
Time to Encrypt One Block (ns)	1233.2	2329	1179.2	2146.8
Time to Decrypt One Block(ns)	1233.2	2329	1179.2	2146.8

Table 3 RC6 3-in-1 Summary

5.2.3 RC6 128 Bit Key Results

Parameter	Iterative 128-Bit		Pipelined 128-Bit	
	Min.	Max.	Min.	Max.
Area (um2)	13,966,637	19,248,830	285,334,063	453,248,223
Transistor Count	217,008	307,247	6,853,619	11,611,314
Input/Outputs Required	520	520	520	520
Throughput (Mbps)	59.68	102.83	1216.38	2196.67
Key Setup Time Encrypt (ns)	5,742.00	13,776.64	3,728.50	6,897.64
Key Setup Time Decrypt (ns)	5,742.00	13,776.64	3,728.50	6,897.64
Algorithm Setup Time (ns)	0	0	0	0
Time to Encrypt One Block (ns)	1,244.80	6,674.62	1,165.40	2,104.60
Time to Decrypt One Block(ns)	1,244.80	6,674.62	1,165.40	2,104.60

Table 4 RC6 128 Bit Key Summary

5.3 RIJNDAEL

5.3.1 Architecture

The following provides a high level description of the major blocks in the RIJNDAEL algorithm. Details of the components, sweeps, and their implementations can be found in the design workbook.

5.3.1.1 Pipelined Key Schedule

The RIJNDAEL key schedule is based on a sliding window approach as described in the algorithm specification. Multiple key sizes are based on the n-1 element and the n-k element (32 bit word organized), where k is 4,6, or 8, depending on key size. The key expansion is a linear combination of the elements, so a similar function can be used on the decrypt function to “unexpand” the keys in a reverse direction. Such an approach allows for an increase in the key agility without sacrificing significant amounts of area to store all of the expanded keys. The encryption expansion can start immediately, with the first words of the initial key being used as expanded key. The setup time for this case is zero. During the decryption, the key is expanded to the last key, stored, and then the pipeline is run to create the previous expanded key until the last decrypt key is generated, which is the initial key. Keys are generated at a rate of four 32 bit words per round, regardless of key size, to keep up with the requirements of the algorithm block. Additional registers are used to maintain sufficient previous keys to generate the next four words of expanded key. Keys are pulled from the bank of registers which make up the sliding window and supplied to the algorithm as sub-keys. S-Boxes are re-used, without a performance penalty, to minimize the size impact of having additional S-Boxes.

5.3.1.2 Pipelined Algorithm

The RIJNDAEL algorithm pipeline consists of a sequential mapping of the steps of the algorithm to registered stages in hardware. Each stage reflects a single round of the algorithm. The primary advantage to pipelining in this manner is the significant increase in throughput. RIJNDAEL was architected such that both the encrypt and decrypt functions could be performed with the same pipeline. This approach needed a static pipeline that could perform both functions, so the algorithm round functions contained in the package will serve a dual role by providing cases for encrypt and decrypt within the same function. The pipeline structure reflects changes in direction, such as requiring a pre-add on the encryption (first round) versus decryption requiring a post-add on the last round.

5.3.1.3 Iterative Key Schedule

The iterative version of the key schedule focuses on reducing the area of the key expansion, so only a single copy of the expansion is maintained. For encryption, as in the pipelined case, the expansion starts immediately, with no key setup required. The keys are expanded every round, producing the four 32 bit words of key required. As each new key is created and stored, the old key is overwritten.

In the case of decryption, the algorithm requires a setup time to effectively run the algorithm to the last key. This serves as the starting point for all decryptions using that key. This value will also be stored so it can be referenced on each new decryption to eliminate key setup for every new decryption.

5.3.1.4 Iterative Algorithm

The algorithm block uses the same functionality as described in the pipeline but does not re-use some of the combination functions used to construct the pipeline. Instead, the function calls are made explicitly, depending on encryption/decryption to provide the widest possible range of hardware re-use. The function calls in the encrypt and decrypt directions are not symmetrical. The algorithm processes the state data on each round, performing only one step of the algorithm per round.

5.3.1.5 Effects of Single Key Sizes

With the RIJNDAEL architecture, the algorithm portion of the design is effected by the key size implementation and will see performance increases across the various implementations. Differences in speed and area will be seen as muxing and control logic can be reduced with single key size implementations. Since RIJNDAEL varies the number of rounds, a proportional reduction in pipelined algorithm area is observed for each of the individual key sizes versus the hybrid (3-in-1) design. Moreover, key size plays a much smaller role in the performance of the key scheduling section. Performance metrics such as area and speed can be isolated in the single key size implementations and measured independently. For the 128 bit key size, the increase in performance across both the lower level functions and higher level upper blocks did not significantly impact the overall performance in speed and area, although there were performance increases. The 3-in-1 design re-used many of the key scheduling logic blocks as possible to reduce overhead, and as a result, only slight differences could be found between the 3-in-1 design and the 128 bit key design. For instance, in the 128-bit key size, there were a larger number of S-boxes required versus individual implementations of the 192-bit and 256-bit key sizes. The amount of overhead associated with a 3-in-1 design did not appear to impact the key schedule design greatly as compared to the 128 bit key size, but for the other sizes, larger differences can be found.

5.3.2 RIJNDAEL 3-in-1 Results

Parameter	Key Size Selected	Iterative 3in1		Pipelined 3in1	
		Min.	Max.	Min.	Max.
Area (um2)	-	24,158,661	46,362,850	307,129,513	471,996,329
Transistor Count	-	481,002	1,029,054	4,190,611	7,130,697
Input/Outputs Required	-	520	520	520	520
Throughput (Mbps)	128 Bit	190.05	447.55	4026.42	5337.78
	192 Bit	158.38	372.96	4026.42	5337.78
	256 Bit	135.75	319.68	4026.42	5337.78
Key Setup Time Encrypt (ns)	128 Bit	0	0	0/29.28*	0/44.76*
	192 Bit	0	0	0/29.28*	0/44.76*
	256 Bit	0	0	0/29.28*	0/44.76*
Key Setup Time Decrypt (ns)	128 Bit	286	673.5	233.99	318.63
	192 Bit	343.2	808.2	262.83	374.33
	256 Bit	400.4	942.9	277.25	402.18
Algorithm Setup Time (ns)	-	0	0	0	0
Time to Encrypt One Block (ns)	128 Bit	286	673.5	239.8	317.9
	192 Bit	343.2	808.2	287.76	381.48
	256 Bit	400.4	942.9	335.72	445.06
Time to Decrypt One Block (ns)	128 Bit	286	673.5	239.8	317.9
	192 Bit	343.2	808.2	287.76	381.48
	256 Bit	400.4	942.9	335.72	445.06

* Note: Both Key Setup/Key Agility times provided, as they are different for this algorithm

Table 5 RIJNDAEL 3-in-1 Summary

5.3.3 RIJNDAEL 128 Bit Key Results

Parameter	Iterative 128-Bit		Pipelined 128-Bit	
	Min.	Max.	Min.	Max.
Area (um2)	20,739,239	33,851,050	262,721,033	419,886,025
Transistor Count	275,485	641,681	3,660,325	4,292,749
Input/Outputs Required	520	520	520	520
Throughput (Mbps)	271.13	605.77	4200.85	5745.06
Key Setup Time Encrypt (ns)	0.00	0.00	0/26.6*	0/33.96*
Key Setup Time Decrypt (ns)	211.30	472.10	226.87	333.26
Algorithm Setup Time (ns)	0	0	0	0
Time to Encrypt One Block (ns)	211.3	472.1	222.8	304.7
Time to Decrypt One Block(ns)	211.3	472.1	222.8	304.7

* Note: Both Key Setup/Key Agility times provided, as they are different for this algorithm

Table 6 - RIJNDAEL 128 Bit Key Summary

5.4 SERPENT

5.4.1 Architecture

The following provides a high level description of the major blocks in the SERPENT algorithm. Details of the components, sweeps, and their implementations can be found in the design workbook.

5.4.1.1 Pipelined Key Schedule

The SERPENT algorithm implements a simple expansion function for the key scheduling. The exclusive-or based function allows for quick computation and does not require key setup in the encrypt direction. Pipelining is maximized as this approach utilizes a sliding window approach, where only a small number of previous expanded keys are needed to compute the next sub-keys. However, for decryption, a key setup time is required to compute the starting point for the key expansion, which is the last set of W registers. The decrypt pipeline computes the previous set of W registers based on the current set, as the exclusive-or based expansion can be reversed easily. To save storage in this design, the keys are computed at each stage, with the decrypt case requiring a block of logic at the beginning to find the last subkeys. The SERPENT pipelined key schedule provides two successive keys to each round of the algorithm on expansion. The algorithm will then select the correct key based on the current encryption/decryption mode. The additional key allows for the rounds that require two keys to operate.

5.4.1.2 Pipelined Algorithm

The pipelined SERPENT algorithm block contains a structural model of the unraveled rounds of the algorithm. Four distinct functions are needed to implement both the encrypt and decrypt operations. The core algorithm round functions are the same for 30 rounds of the algorithm, with an internal mux/demux to select the encrypt or decrypt mode. The first two rounds of encrypt and last two rounds of decrypt distinguish the cases where the pipeline is re-routed. The encrypt will bypass the two special rounds of the decrypt while the decrypt will bypass the two special rounds of encrypt. The latency will remain the same as no extra rounds are added. The pipeline will select and re-route based on the current mode of encryption or decryption.

5.4.1.3 Iterative Key Schedule

The SERPENT iterative key schedule uses a single copy of the expansion function to generate the sub-keys, one at a time. Area can be significantly reduced by using the same hardware repeatedly. Additional key setup will be required in the decrypt direction to allow for the run-up to the last key of the expansion.

5.4.1.4 Iterative Algorithm

The iterative algorithm uses the same functions as the pipeline, with the same round instance referenced repeatedly to perform the main processing of the algorithm. The special case rounds are selected by the state machine within the iterative block to determine encrypt/decrypt direction, and consequently, which pre/post add functions to perform.

5.4.1.5 Effects of Single Key Sizes

Within the implementation of the SERPENT algorithm, the key size is padded to 256 bits internally, so the effects of varying key sizes have negligible impact on the performance metrics of the design. Consequently, only the 3-in-1 cases is provided.

5.4.1.6 Key Setup Reduction

An alternative method of calculating the last 8 32-bit words of the decrypt key schedule has been offered as a viable alternative to running the full encrypt initial expand function to extract these keys. It was noted by several individuals at the third AES conference that the last 8 words of the key expansion could be determined by exclusive OR'ing key bits of the initial CV and effectively jumping to the last keys in a single step. Not only is this much faster, but surprisingly it comprises less area than the initial expand function which was reported on earlier. The table below compares this decrypt modification to the initial expand function

Approach	Best Case Timing	Area
Original (Unrolled method)	212.55	16,856,672
Improved (Single Step)	6.95	12,901,915

Table 7 - SERPENT Key Setup Reduction Summary

5.4.2 SERPENT 3-in-1 Results

Parameter	Iterative 3in1		Pipelined 3in1	
	Min.	Max.	Min.	Max.
Area (um2)	13,779,834	23,274,086	210,124,852	438,561,500
Transistor Count	204,617	345,483	3,298,104	5,741,469
Input/Outputs Required	520	520	520	520
Throughput (Mbps)	102.00	202.30	5298.01	8030.11
Key Setup Time Encrypt (ns)	19.77	39.21	18.98	22.91
Key Setup Time Decrypt (ns)	672.18	1333.14	212.55	365.58
Algorithm Setup Time (ns)	0	0	0	0
Time to Encrypt One Block (ns)	510.08	775.18	510.08	773.12
Time to Decrypt One Block(ns)	510.08	775.18	510.08	773.12

Table 8 SERPENT Summary

5.5 TWOFISH

The following provides a high level description of the major blocks in the TWOFISH algorithm. Details of the components, sweeps, and their implementations can be found in the design workbook.

5.5.1 Architecture

A useful property of the TWOFISH architecture was the relatively large amount of design block re-use. Both the Key Schedule and Algorithm utilized many of the same functions. While this does not result directly in a direct increase in performance, since key expansion and encryption are performed in parallel, it does simplify the hardware coding process. As stated, common coding techniques and processes were used for developing each algorithm design resulting in areas available for improvement in a more highly optimized design. In the case of TWOFISH, a smaller design could be created by taking advantage of the function re-use. However, as with most hardware trade-offs, this area optimization would come at the expense of performance and complex control mechanisms. Another feature of TWOFISH is the lack of initial key runup prior to subkey expansion. In addition, the key schedule is not a feed-forward design. Each round of key schedule is independent of the previous round. This unique characteristic allowed for a Key Schedule that does not require a setup time for either encryption or decryption.

In the TWOFISH algorithm, the first step of encryption is a pre-whiten function, In hardware, this is simply an exclusive-OR. The pre-whiten step is performed during the same clock cycle as the first subkey expansion which generates the pre-whiten subkey. This was possible because the XOR function did not create a critical path concern since the main algorithm rounds incorporate an integer addition that is more complex. The result was the ability to load data and key in the same clock cycle, thereby reducing the overall time for encryption by one clock cycle.

5.5.1.1 Pipelined Key Schedule

In order to allow for either encryption or decryption, both pre-add and post-add subkeys are generated during the first pipeline stage. The post-add key is buffered through a pipelined delay until needed in the final processing step. Since one of the input parameters is the round number which is fixed for a given pipelined round there is an optimization or pre-calculation in each pipelined round.

5.5.1.2 Pipelined Algorithm

The algorithm is an efficient unrolling of stages because encryption and decryption are nearly identical. In addition, the symmetry allows for similar processing in both the encrypt and decrypt directions.

5.5.1.3 Iterative Key Schedule

As in the pipelined key schedule, the iterative design requires buffering of the post-add subkey until it is needed in the final processing step. However, this buffering is not required to be implemented in pipelined stages. The key schedule round is generalized such that the round number is not a fixed constant as in the pipelined case. This does not allow synthesis optimization of each round, but does save area since only one hardware block is instantiated.

5.5.1.4 Iterative Algorithm

The differences between encryption and decryption are minor such that the additional hardware to support either process in a single round adds minimal area.

5.5.1.5 Effects of Single Key Sizes

The performance of TWOFISH changes with key size. The permutations in both the key schedule round and algorithm round depend on the key size. There is one additional permutation function for 192-bit keys versus the 128-bit keys, and two additional permutation functions for 256-bit keys. The effect is increasing the critical path for longer key sizes. Therefore, the timing and area increase with key size, and performance is decreased slightly. The performance of the 3-in-1 implementation is comparable to the 256-bit key implementation since the critical path is identical with the exception of some multiplexing to switch in data of smaller key sizes. A feature of the 3-in-1 hardware implementation is the ability to over-clock the same piece of hardware in cases where only 128-bit keys are being used since in such cases the critical path is actually shorter than the worst-case path (256-bit key case). The timing and throughput numbers of the 3-in-1 implementation reported here use this feature.

5.5.2 TWOFISH 3-in-1 Results

Parameter	Key Size Selected	Iterative 3in1		Pipelined 3in1	
		Min.	Max.	Min.	Max.
Area (um2)	-	12,700,019	23,044,514	187,058,851	343,109,723
Transistor Count	-	182,533	377,599	2,666,940	5,750,915
Input/Outputs Required	-	520	520	520	520
Throughput (Mbps)	128 Bit	59.50	104.60	1429.00	2278.00
	192 Bit	50.00	90.96	1164.00	1957.00
	256 Bit	42.95	80.51	977.20	1718.00
Key Setup Time Encrypt (ns)	128 Bit	42.48	65.36	0/65.63*	0/127.42*
	192 Bit	61.28	106.78	0/65.63*	0/127.42*
	256 Bit	79.49	149	0/65.63*	0/127.42*
Key Setup Time Decrypt (ns)	128 Bit	42.48	65.36	0/65.63*	0/127.42*
	192 Bit	61.28	106.78	0/65.63*	0/127.42*
	256 Bit	79.49	149	0/65.63*	0/127.42*
Algorithm Setup Time (ns)	-	0	0	0	0
Time to Encrypt One Block (ns)	128 Bit	1223.2	2151.2	1123.8	1791
	192 Bit	1407.2	2559.6	1307.8	2199.4
	256 Bit	1589.8	2980	1490.4	2619.8
Time to Decrypt One Block (ns)	128 Bit	1223.2	1223.2	1123.8	1791
	192 Bit	1407.2	1407.2	1307.8	2199.4
	256 Bit	1589.8	1589.8	1490.4	2619.8

* Note: Both Key Setup/Key Agility times provided, as they are different for this algorithm

Table 9 TWOFISH Summary

5.5.3 TWOFISH 128 Bit Key Results

Parameter	Iterative 128-Bit		Pipelined 128-Bit	
	Min.	Max.	Min.	Max.
Area (um2)	9,158,239	16,110,756	121,705,687	225,298,323
Transistor Count	134,997	264,058	1,785,286	3,783,973
Input/Outputs Required	520	520	520	520
Throughput (Mbps)	60.84	105.14	1341.44	2273.53
Key Setup Time Encrypt (ns)	60.87	105.2	0/47.29*	0/91.86*
Key Setup Time Decrypt (ns)	60.87	105.2	0/47.29*	0/91.86*
Algorithm Setup Time (ns)	0	0	0	0
Time to Encrypt One Block (ns)	1217.4	2104.00	1126	1908.4
Time to Decrypt One Block(ns)	1217.4	2104.00	1126	1908.4

* Note: Both Key Setup/Key Agility times provided, as they are different for this algorithm

Table 10 - Twofish 128 Bit Key Summary

6 Performance Results – 3-in-1 Design

6.1 Summary

A table summarizing the results and performance metrics is given below for algorithm comparison. These comparison values are given only for the combined key size design, which implements a selectable 128-bit, 192-bit, or 256-bit key in the same device.

Parameter	Algorithm				
	MARS	RIJNDAEL	RC6	SERPENT	TWOFISH
Area (um2)	127,432,766	46,361,993	21,660,006	23,274,086	23,044,514
Transistor Count	1,950,277	1,029,046	430,436	345,483	377,599
Input/Outputs Required	520	520	520	520	520
Throughput (Mbps)	56.7	443.2	103.8	202.3	104.6
Key Setup Time Encrypt (ns)	9553	0	8139	19.77	61.16
Key Setup Time Decrypt (ns)	27470	288.8	8139	672.18	61.16
Algorithm Setup Time (ns)	0	0	0	0	0
Time to Encrypt One Block (ns)	2256.9	288.8	1233.2	632.6	1223.2
Time to Decrypt One Block(ns)	2256.9	288.8	1233.2	632.6	1223.2

Table 11 Iterated Summary

Parameter	Algorithm				
	MARS	RIJNDAEL	RC6	SERPENT	TWOFISH
Area (um2)	1,333,099,627	471,996,329	554,268,739	438,561,500	343,109,723
Transistor Count	20,667,308	7,130,697	9,013,872	5,741,469	5,750,915
Input/Outputs Required	520	520	520	520	520
Throughput (Mbps)	2189	5163	2171	8030	2278
Key Setup Time Encrypt (ns)	3718	0	3660	18.98	0
Key Setup Time Decrypt (ns)	3718	233.99	3660	212.55	0
Algorithm Setup Time (ns)	0	0	0	0	0
Time to Encrypt One Block (ns)	1988	248	1179	510	1123
Time to Decrypt One Block(ns)	1988	248	1179	510	1123

Table 12 Pipelined Summary

6.2 Performance Comparisons

6.2.1 Iterative Area

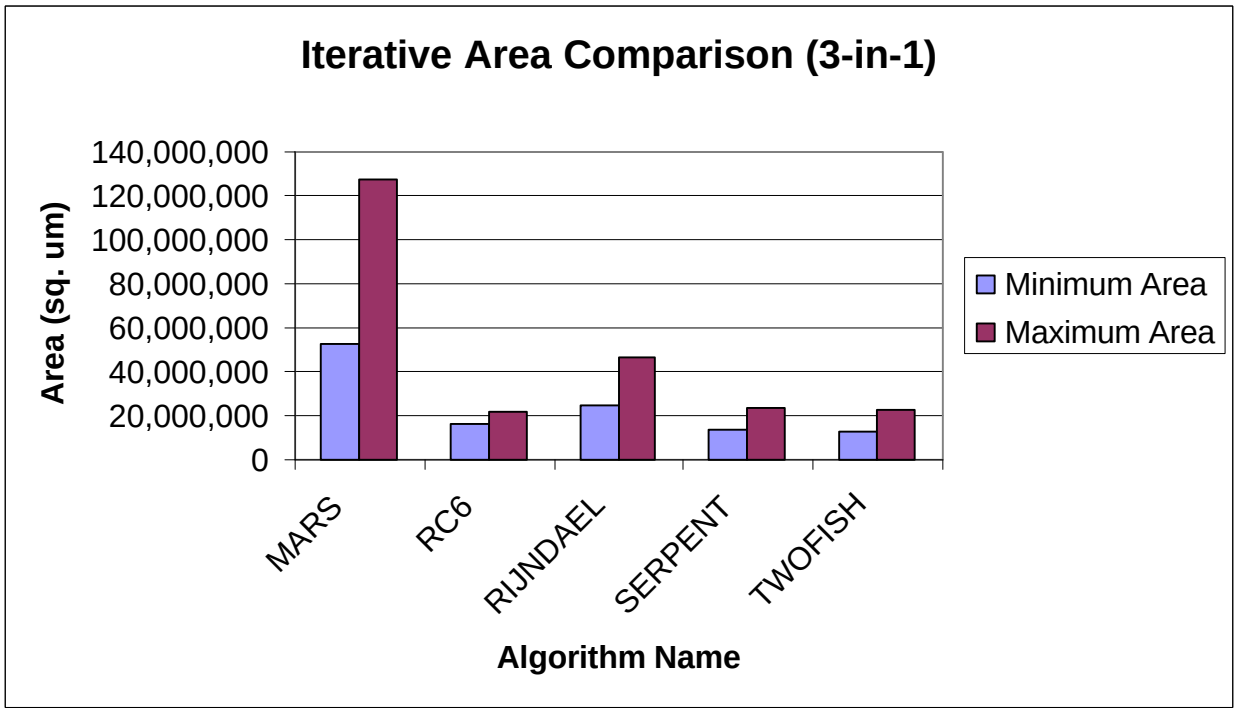

Figure 4 Iterative Comparison – 3-in-1 Area

From the graph, one can see that the RC6 algorithm provides the smallest area in the fastest configuration of the iterative design (maximum area), but TWOFISH provides the smallest device overall (minimum area). In the iterative case, RIJNDAEL provides the highest performance in terms of throughput, but as evidenced by the graph, incurs a slightly higher area penalty. All numbers shown are total area and consist of the algorithm, key schedule, interface, and controller blocks combined.

6.2.2 Iterative Transistor Count

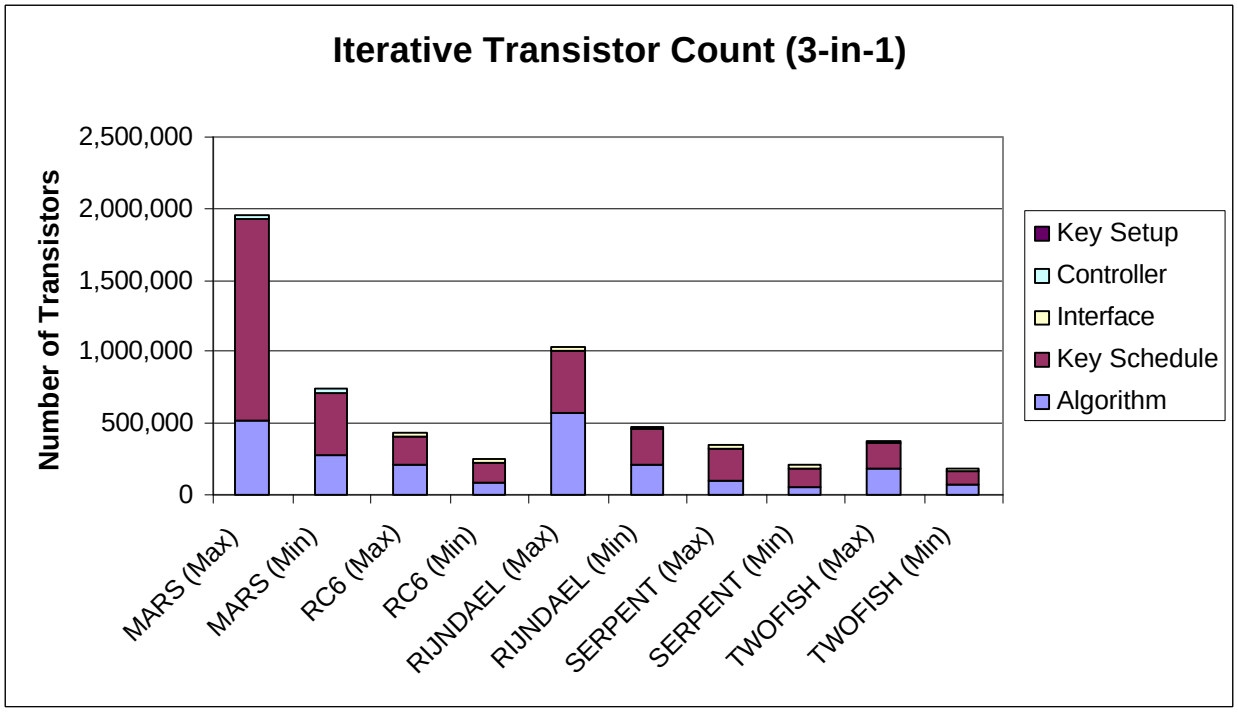

Figure 5 - Iterative 3-in-1 Transistor Count

Figure 5 provides the total number of transistors required to build the algorithms in the iterative design. A minimum and maximum version are provided to show the range of the architectures. Also provided are breakdowns by design block to provide insight as to the size required for some of the major components. Clearly, the algorithm and key schedule blocks comprise the majority of each design. In addition, the key schedule is often larger than the algorithm. The smallest design in the iterative case is the TWOFISH algorithm, with RC6 and SERPENT only slightly larger. The largest design is MARS especially evident in the maximum (largest area and throughput) version. RIJNDAEL requires a slightly higher transistor count than the low three (TWOFISH, RC6 and SERPENT), but as the following graphs will indicate, there is a significant increase in the maximum throughput achieved.

6.2.3 Iterative Throughput

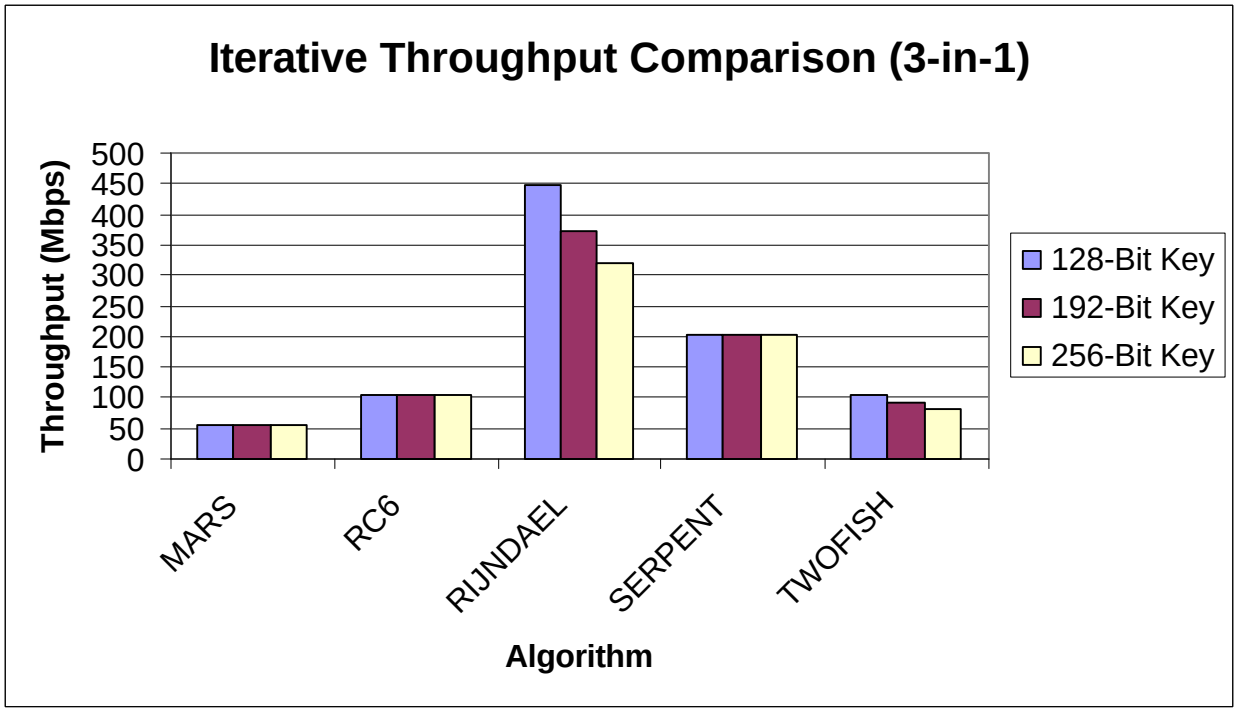

Figure 6 3-in-1 Iterative Comparison - Throughput

From Figure 6, one can see that the RIJNDAEL algorithm provides the highest level of throughput for all of the algorithms in the iterative case; its throughput ranges from 316 to 443 Mbps depending on the key size. SERPENT follows with a throughput of about half that of RIJNDAEL, and provides 202 Mbps. The remaining algorithms fall into the range of 60 to 100Mbps. Although not shown on this graph, the minimum clock period for SERPENT is actually smaller than RIJNDAEL, but for RIJNDAEL, the reduced number of rounds (as compared to SERPENT) significantly impacts the throughput achieved.

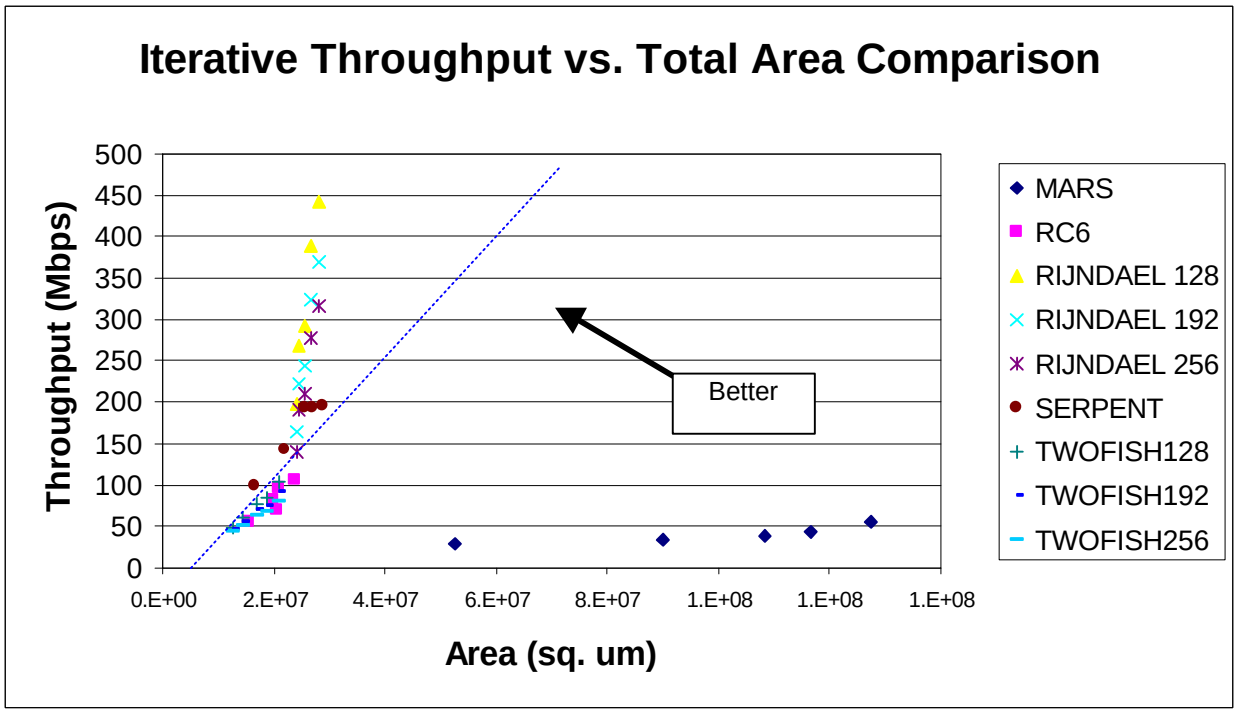

Figure 7 - Iterative Throughput vs. Area Comparison

Figure 7 describes the relationship and trade-offs between the throughput performance versus the amount of hardware area required to attain that level of performance. In this graph, the optimal place to be is the upper left corner, where throughput is maximized, while area is minimized. The graph shows that the majority of the algorithms follow a fairly linear trend. For an increase in area, there is a corresponding increase in throughput. On the higher end, it can be noted that RIJNDAEL achieves the highest throughput performance. More significantly, its increase in throughput is achieved at a smaller area increment relative to the other algorithms. Boosting the throughput for RIJNDAEL does not appear to have as significant an impact on area as the other algorithms. On the lower end, MARS has a smaller performance improvement compared to area savings; the percent change in area is greater than the percent change in throughput. Specifically, for MARS a 59% increase in area yields a throughput increase of only 49% whereas a 13% increase in RIJNDAEL area yields a 56% increase in throughput. The remaining algorithms (RC6, TWOFISH and SERPENT) achieve a slightly greater throughput increase than cost of area increase.

To elaborate on the performance and area trade-offs, the following table summarizes the expected throughput per unit area.

Algorithm	Configuration					%Change (high-low)
	Speed 1	Speed 2	Speed 3	Speed 4	Speed 5	
MARS	0.45	0.38	0.37	0.38	0.55	52%
RC6	4.49	4.58	4.16	3.39	3.55	35%
RIJNDAEL128	15.89	14.60	11.41	10.98	8.16	95%
RIJNDAEL192	13.24	12.17	9.51	9.15	6.80	95%
RIJNDAEL256	11.35	10.43	8.15	7.85	5.83	95%
SERPENT	6.80	7.15	7.52	6.48	5.89	28%
TWOFISH128	4.98	4.52	4.51	4.09	3.86	29%
TWOFISH192	4.33	4.02	4.06	3.77	3.60	20%
TWOFISH256	3.83	3.63	3.70	3.49	3.38	13%

Table 13 – 3-in-1Throughput Bits per Unit Area (in bits/sq. um)

In Table 13, each of the algorithms are represented for the 3-in-1 case, and when varying modes are available (e.g., RIJNDAEL, TWOFISH), throughputs are included for the 3-in-1 design running in each of the key sizes. The configurations represent the varying speed grades of the design, where speed 1 is the fastest, highest area design. Conversely, speed5 is the slowest, least area intensive design. Each entry represents the amount of throughput (in bits/sec) that an algorithm will return given a 1 square micron unit of area. RIJNDAEL provides the highest throughput per area and also the greatest increase in throughput from the low speed to high speed, with an increase of 95%.

6.2.4 Iterative Key Setup Time (Encrypt)

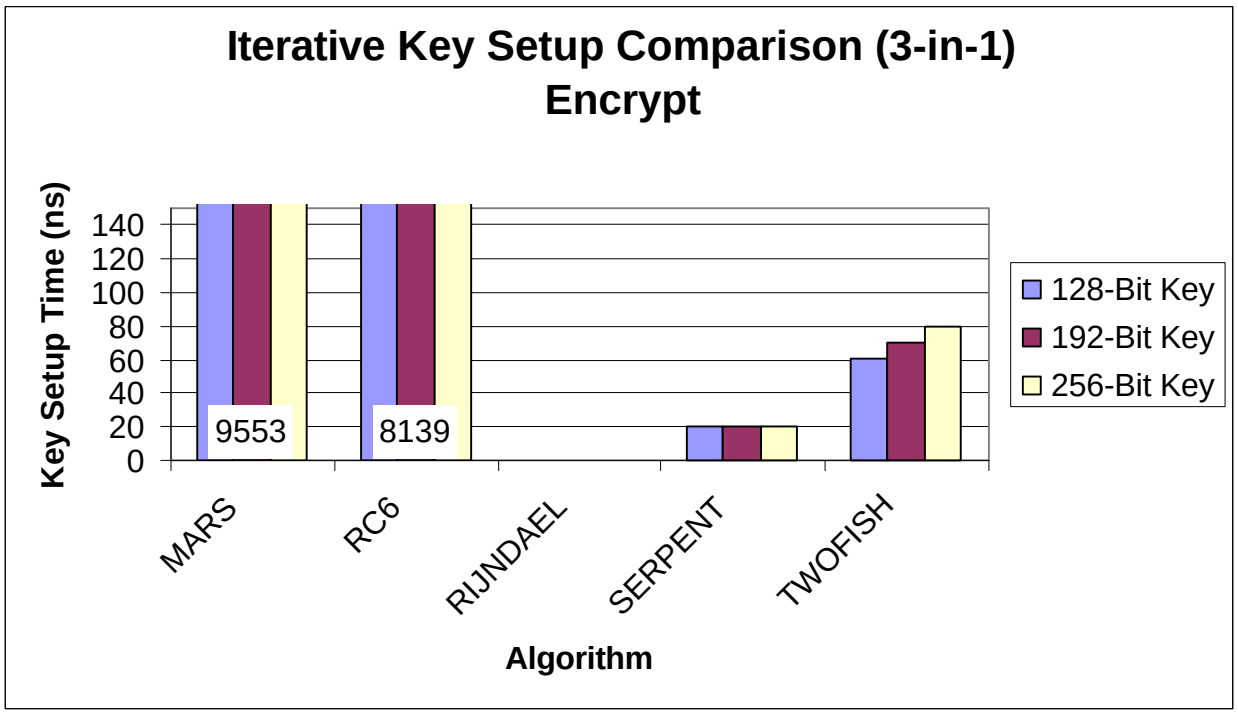

Figure 8 - 3-in-1 Iterative Key Setup (Encrypt)

From Figure 8, it is evident that there is a wide range in the amount of time required for key setup. The highest times fall mainly with RC6 and MARS, with both being shown on the graph as off-scale. While the RC6 and MARS key expansion schedules are very efficient, they require a complex initial key setup thus reducing the key agility. RIJNDAEL uses the user supplied key directly in the first round of the algorithm, thus, there is no key setup time in the encrypt direction. SERPENT has a very minimal key setup time for encryption as well. The TWOFISH key setup time is relatively significant in the iterative case since minimum area is the primary constraint. Therefore, a single key schedule round (which produces 64 bits of subkey) is implemented, and data must be passed through this round twice in order to produce the full 128 bits required in the first round pre-whiten stage.

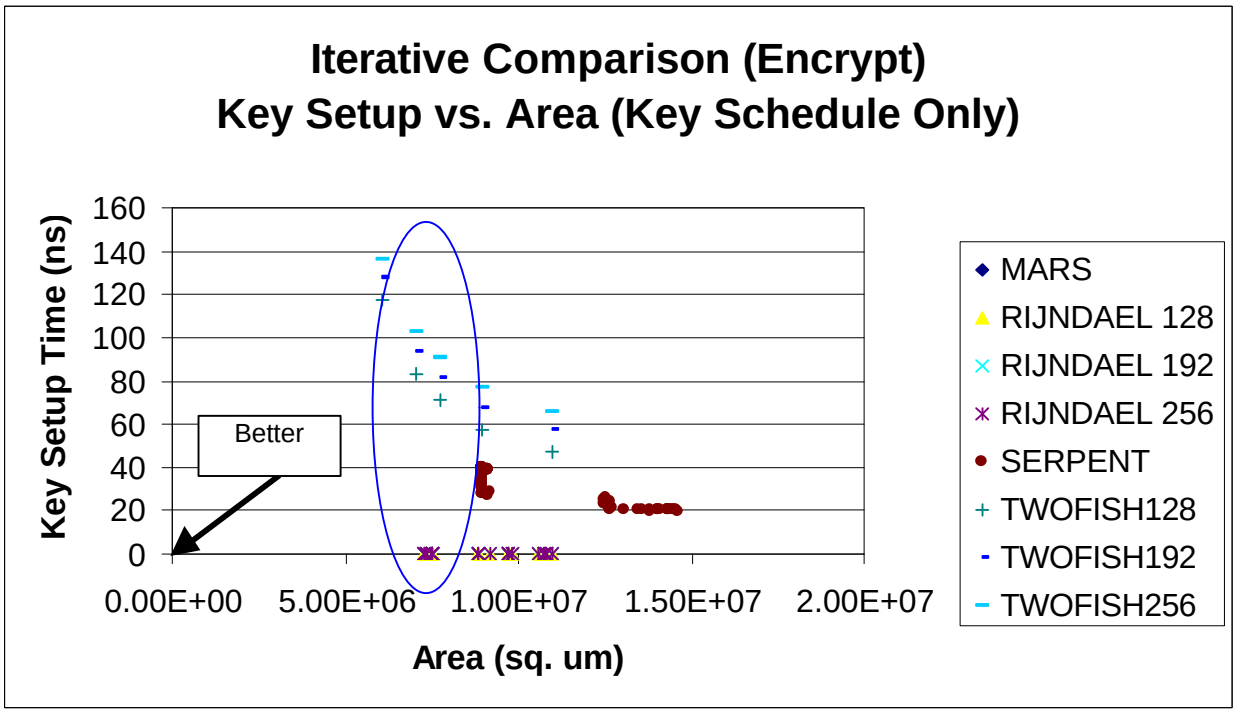

Figure 9 - Iterative Key Setup vs. Area (Encrypt)

Figure 9 combines both the key setup time and iterative area into a single graph for analysis. In this instance, the area represented is the hardware area required by the key schedule only, and does not contain the remaining blocks in the design. The optimal place in this graph is in the lower left corner, with key setup time and area both minimized. Both MARS and RC6 are off the scale of the graph, and as a result, are not shown. The circled region highlights the minimal area for all algorithms' key schedules, and shows the relative performance of the RIJNDAEL and TWOFISH, the two smallest iterative key schedules. RIJNDAEL provides a zero setup time in the encrypt direction, but as the graph indicates, provides this significant performance at a slightly increased area cost over the smallest TWOFISH implementations. While even the smallest SERPENT implementations require more area, the key setup time is smaller than TWOFISH.

6.2.5 Iterative Key Setup Time (Decrypt)

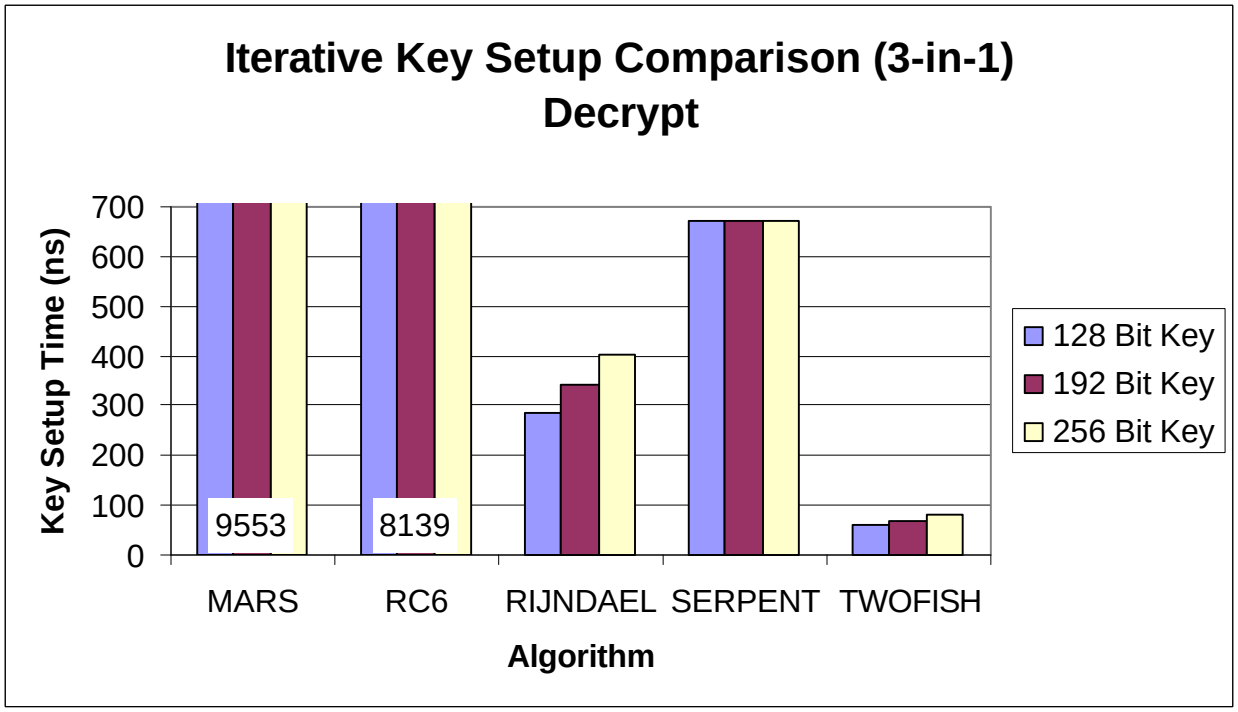

Figure 10 - 3-in-1 Iterative Key Setup Comparison

Figure 10 shows the dependence of key setup time on the direction of the algorithm processing. TWOFISH, RC6, MARS remain constant in either encrypt or decrypt directions. In the decrypt direction, the last subkeys (relative to encrypt subkey order) must be generated prior to beginning the decryption process. RIJNDAEL and SERPENT are adversely impacted since their key schedules are derived from a feed-forward architecture in which the next subkey depends on the previous subkey thus requiring the entire key expansion to be executed prior to decryption. This greatly increases the key setup times and reduces the key agility in the decrypt direction. A transformation is possible for SERPENT which can significantly reduce this decrypt setup time, and a brief discussion and estimates are provided in 5.4.1

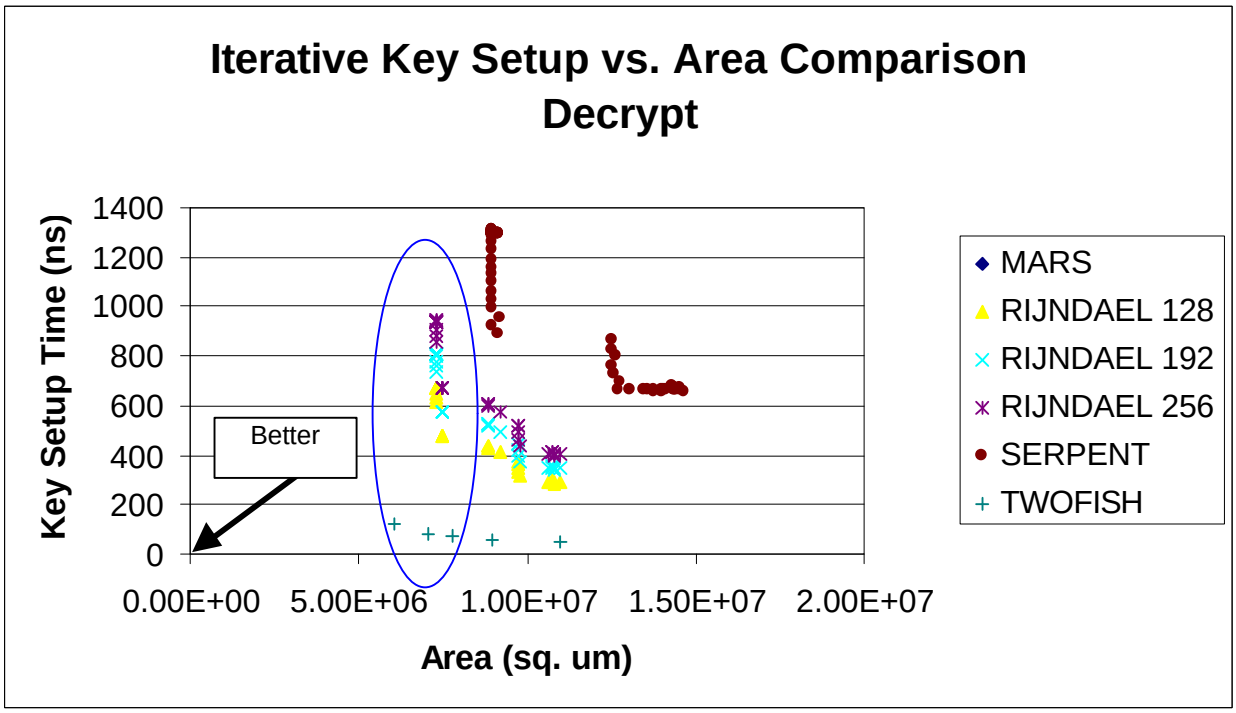

Figure 11 - Iterative Key Setup vs. Area (Decrypt)

As in Figure 9, this graph combines the key setup time and area performance metrics, but examines the decrypt direction. In the cases where all of the keys are not generated beforehand, algorithms will have to create the last subkeys before processing can begin. It's interesting to note the effect that the decrypt key setup has on several of the algorithms. Both RIJNDAEL and SERPENT setup times are increased significantly over their corresponding encrypt direction. TWOFISH remains constant in both directions and switches easily between the two. With the same area highlighted as in Figure 9, the dramatic increase in key setup can be observed. However, further analysis with the SERPENT key scheduling has introduced a transformation which can significantly decrease the setup time. Details are provided in 5.4.1

6.2.6 Iterative Time to Encrypt

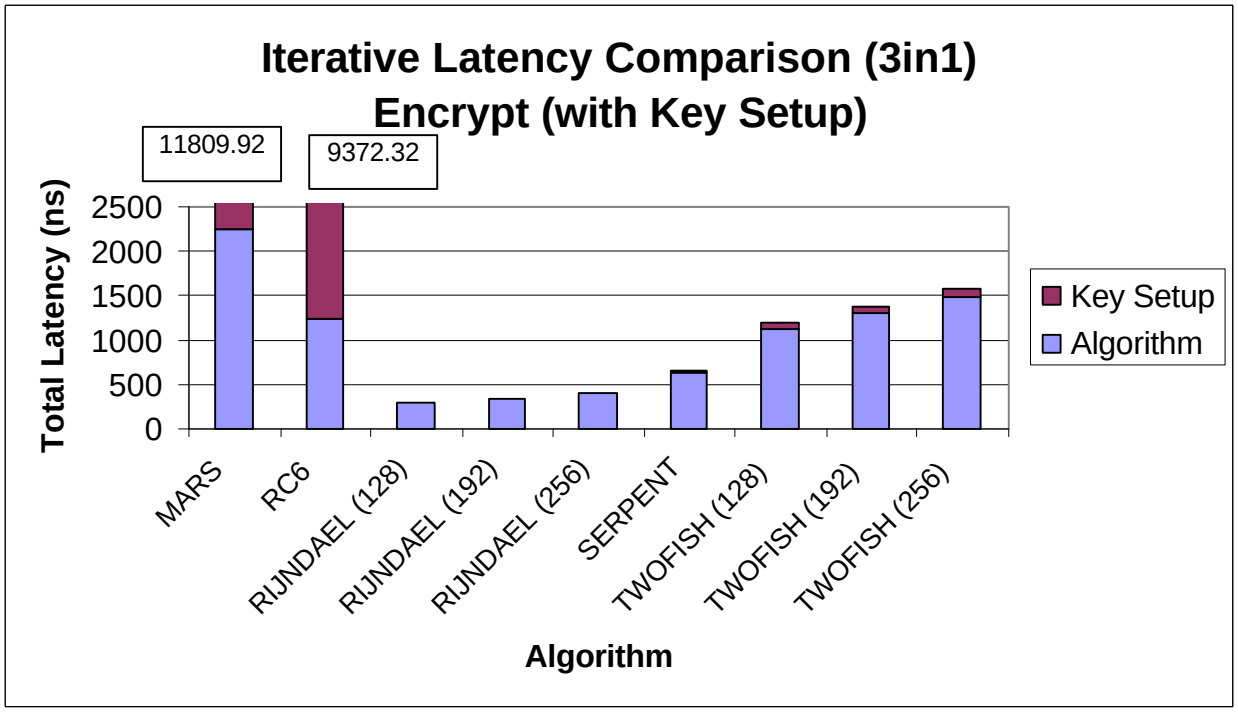

Figure 12 - Iterative 3-in-1 Latency Comparison (Encrypt)

Figure 12 provides details on the latency of each of the algorithms in the encrypt direction. The total latency considers the time to setup the key as well as the time to encrypt. The algorithm portion represents the time, in nanoseconds, that is required to encrypt a block of data. For algorithms that have key setup, the time required for establishing subkeys used in the encryption is provided as well. RIJNDAEL has the minimum latency of all algorithms of 288 ns for 128-bit keys (404 ns for 256-bit keys). RC6 and MARS have latency values that are off the scale of this figure due to the lengthy key setup time.

6.2.7 Iterative Time to Decrypt

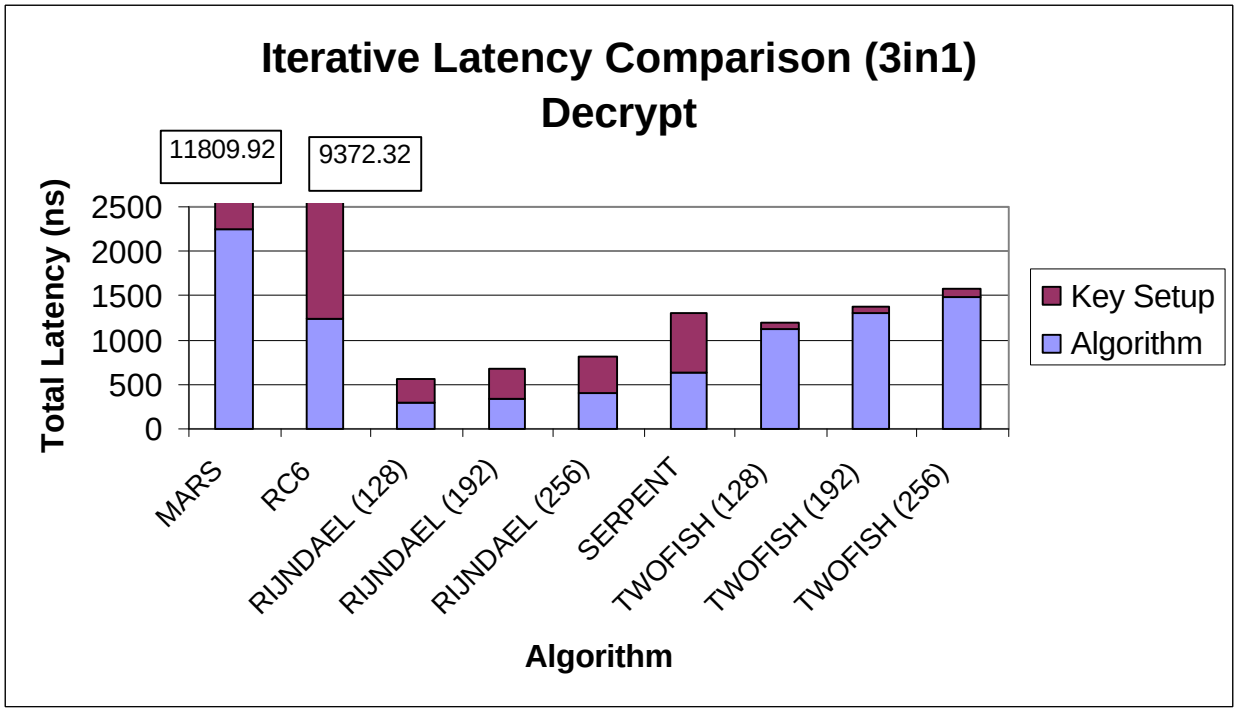

Figure 13 - Iterative 3-in-1 Latency Comparison (Decrypt)

Figure 13 shows a similar result as Figure 12, but in the decrypt case the effects of key setup can be seen more readily. RIJNDAEL provides the smallest latency with its reduced number of rounds and fast round time. Even with the decrypt key setup included, RIJNDAEL still provides the lowest latency. As SERPENT provided a runner-up to RIJNDAEL in the encrypt direction, it can be seen that the increased latency of the key setup puts SERPENT in the mid-range of TWOFISH for overall latency. However, with the transformation that can be applied to SERPENT's key setup, the overall total will once again fall below TWOFISH in overall latency.

6.2.8 Pipelined Area

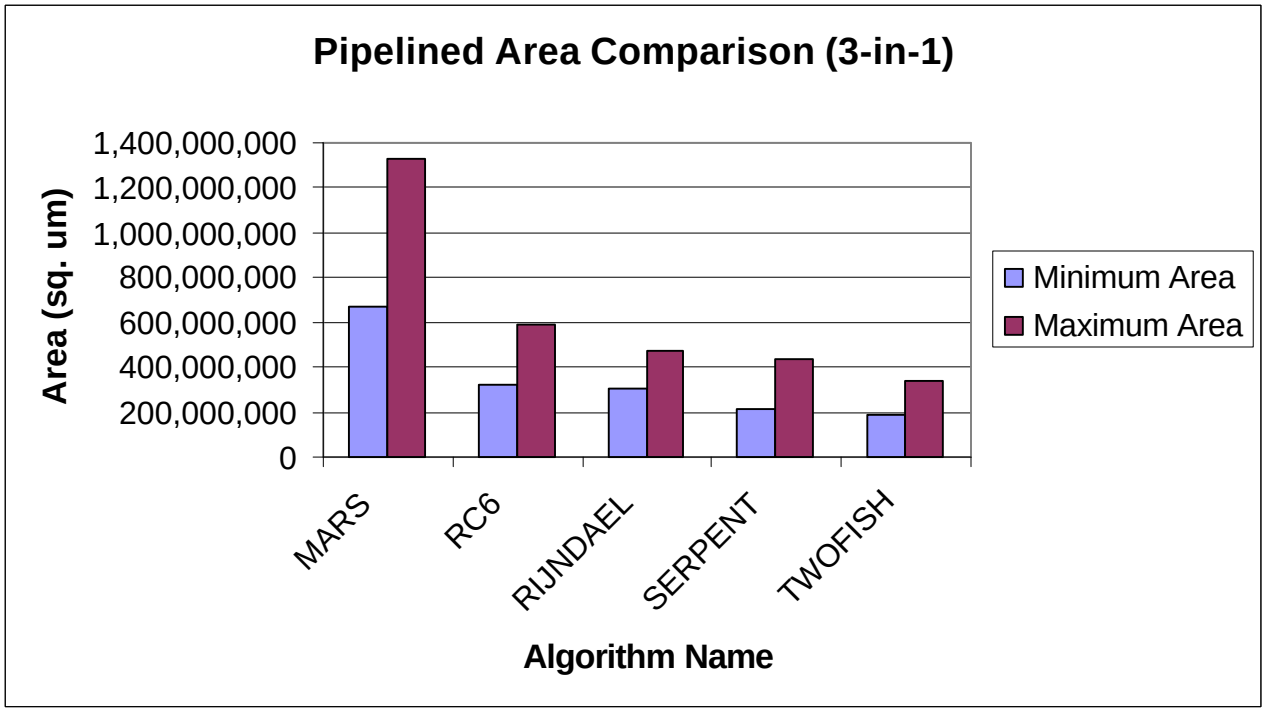

Figure 14 - Pipelined 3-in-1 Area Comparison

Figure 14 demonstrates the maximum and minimum areas attained for each of the pipelined algorithms. The minimum area represents the point where further area reduction was not achievable by the synthesis tool whereas the maximum area represents the point where further increases in area did not result in a faster design. The majority of the algorithms fall into the 400-500M sq. um range, with TWOFISH being the smallest design. MARS did not follow the range and is significantly higher in this metric.

6.2.9 Pipelined Transistor Count

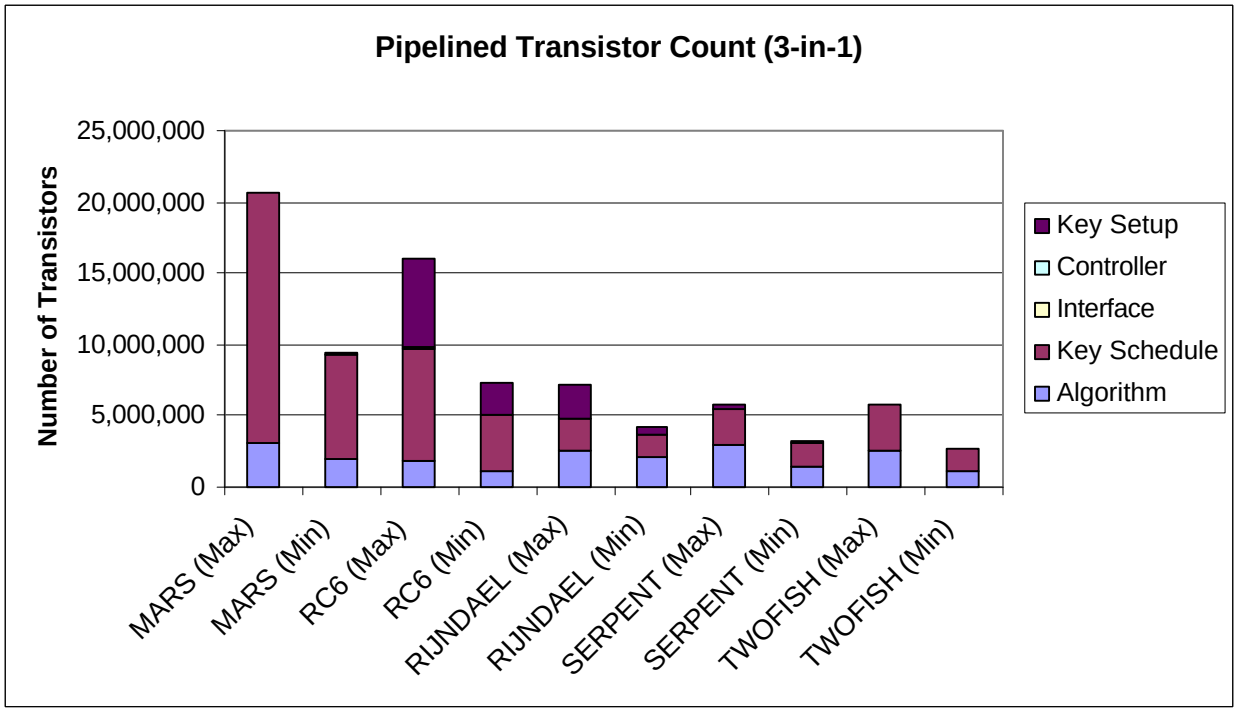

Figure 15 - Pipelined 3-in-1 Transistor Count

Figure 15 shows the minimum and maximum transistor counts required to implement the pipelined algorithms. Also included are the breakdowns by design block. In this case, an additional block, named key setup, is provided to show the percentage of transistor count dedicated to key setup in the cases where large size required a separate architecture block.

6.2.10 Pipelined Throughput

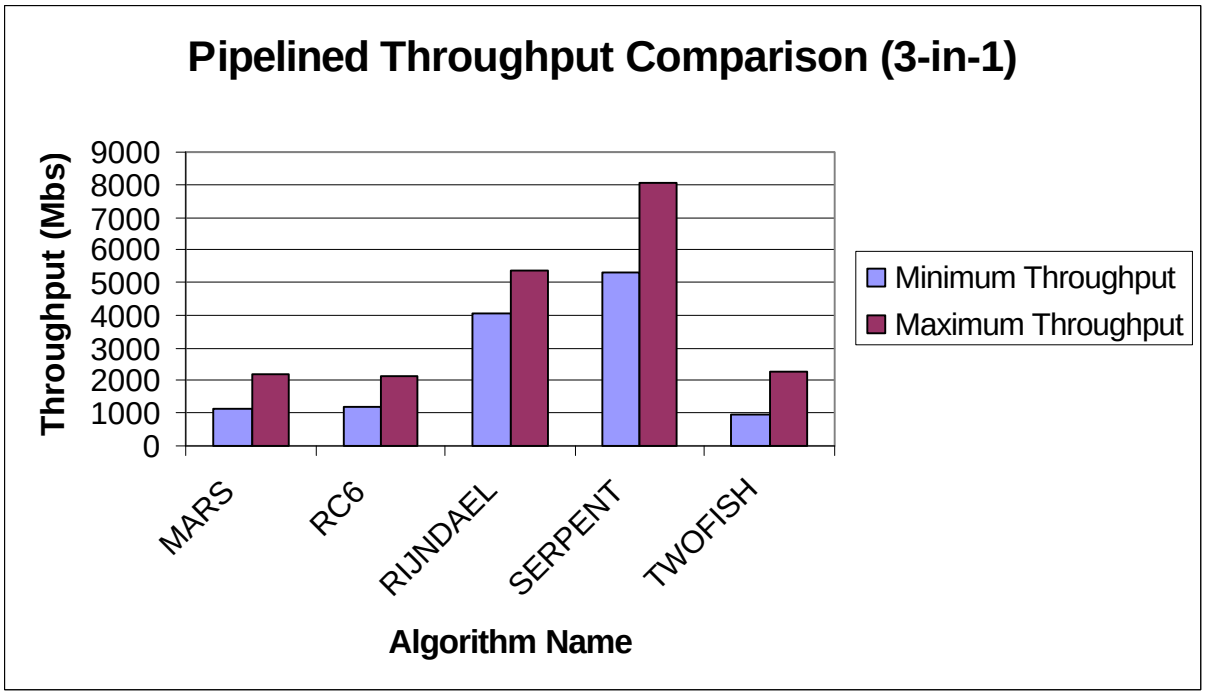

Figure 16 - 3-in-1 Pipelined Throughput Comparison

Figure 16 describes the range of throughput values attainable for each of the algorithms. The maximum throughput represents the pipelined case in its largest and fastest configuration. The minimum throughput is also a pipelined case, but represents the throughput using the smallest configuration of the pipeline. SERPENT achieves the greatest throughput at 8.03 Gbps. RIJNDAEL is approximately 35% slower at 5.16 Gbps. RC6, TWOFISH and MARS are within 5% of each other ranging from 2.17 Gbps (RC6) to 2.28 Gbps (TWOFISH).

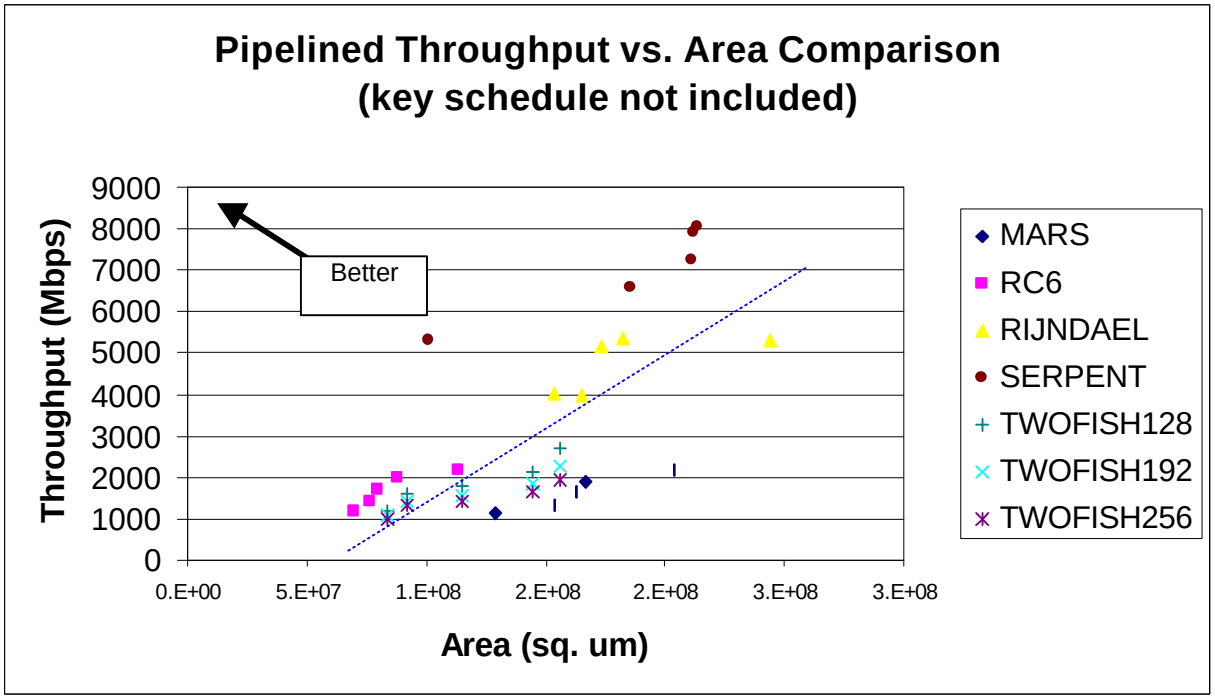

Figure 17 - Pipelined 3-in-1 Throughput vs. Area Comparison

Figure 17 displays the relationship between the pipelined throughput and the area of the algorithm portion of the design. In this case, the better performance is realized in the upper left corner of the graph, minimizing area and maximizing throughput. A dotted line is shown to show the rough linear correlation between the amount of resources (i.e, area) required for a given throughput level. In this case, considering the areas of the algorithm portions to be approximately equal, SERPENT provides the highest level of performance followed by RIJNDAEL.

To elaborate on the performance and area trade-offs, the following table summarizes the expected throughput per unit area.

Algorithm	Configuration					%Change
	Speed 1	Speed 2	Speed 3	Speed 4	Speed 5	(high-low)
MARS	10.75	11.24	10.24	8.72	8.73	29%
RC6	19.08	22.51	21.40	18.56	17.15	31%
RIJNDAEL	21.14	28.60	27.12	24.93	26.36	35%
SERPENT	37.52	37.24	34.19	35.36	52.59	54%
TWOFISH128	17.30	14.72	15.57	17.37	14.04	24%
TWOFISH192	14.48	12.77	13.79	15.58	12.95	22%
TWOFISH256	12.47	11.28	12.39	14.13	12.02	25%

Table 14 - 3-in-1 Throughput Bits Per Unit Area (bps/sq. um)

In, each of the algorithms are represented for the 3-in-1 case, and when varying modes are available (e.g., RIJNDAEL, TWOFISH), throughputs are included for the 3-in-1 design running in each of the key sizes. The configurations represent the varying speed grades of the design, where speed 1 is the fastest, highest area design. Conversely, speed5 is the slowest, least area intensive design. Each entry represents the amount of throughput (in bits/sec) that an algorithm will return given a 1 square micron unit of area. In this pipelined case, SERPENT provides the highest throughput per area and also the greatest increase in throughput from the low speed to high speed, with an increase of 54%.

6.2.11 Pipelined Key Setup Time (Encrypt)

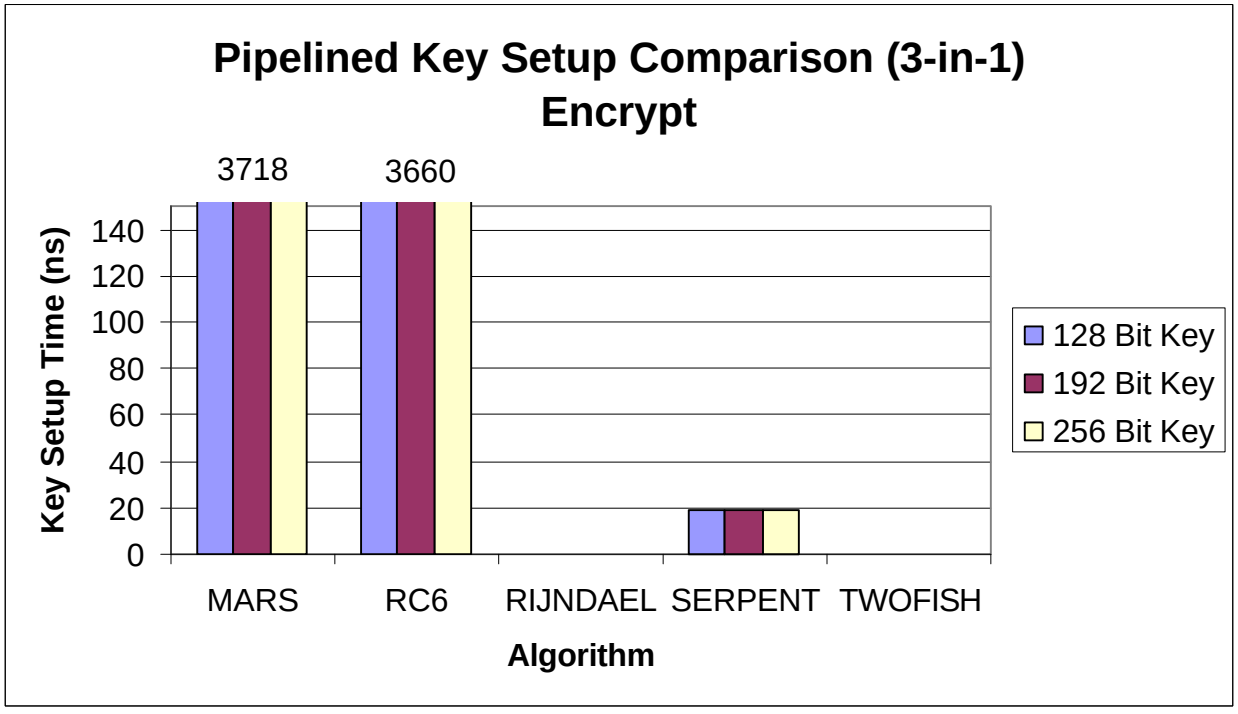

Figure 18 - Pipelined 3-in-1 Key Setup Comparison (Encrypt)

Figure 18 shows the key setup times required by the algorithms in the pipelined case. Note in this case RIJNDAEL and TWOFISH both produce zero setup times. The pipelined implementation of TWOFISH allows for parallel key run-up blocks, and as a result, eliminates the setup time through shared hardware resources. SERPENT follows with minimal key setup time. While pipelining methods were able to reduce the overall key setup times, MARS and RC6 still have excessive key setup times relative to RIJNDAEL, TWOFISH and SERPENT.

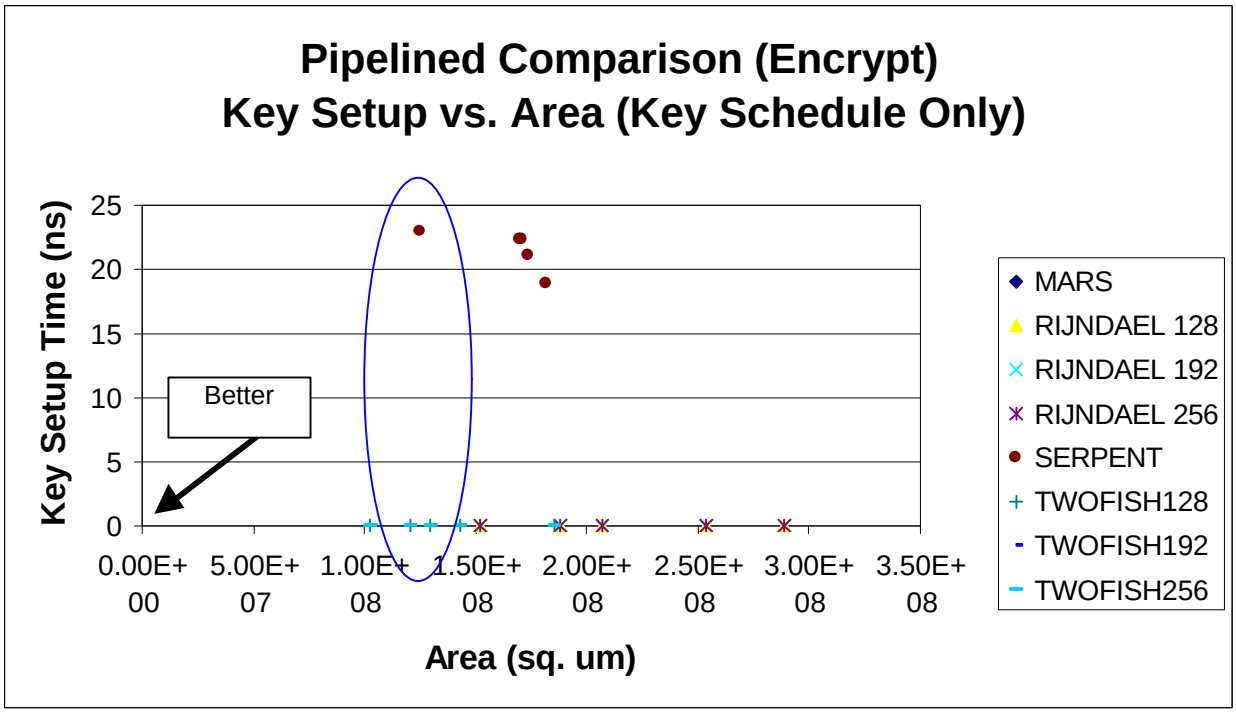

Figure 19 - Pipelined 3-in-1 Key Setup vs. Area (Encrypt)

Figure 19 describes the relationship between encrypt key setup time and the amount of hardware area required to implement the performance listed. For this case, MARS and RC6 are off of the scale, with the remaining algorithms shown for various modes of the 3-in-1 design. TWOFISH and RIJNDAEL provide the minimal amount key setup time at zero, but overall, TWOFISH obtains the performance with less hardware resources. Similar performance in terms of area can be seen with SERPENT, but as the circled area shows, there is a slight increase in setup time required.

6.2.12 Pipelined Key Setup Time (Decrypt)

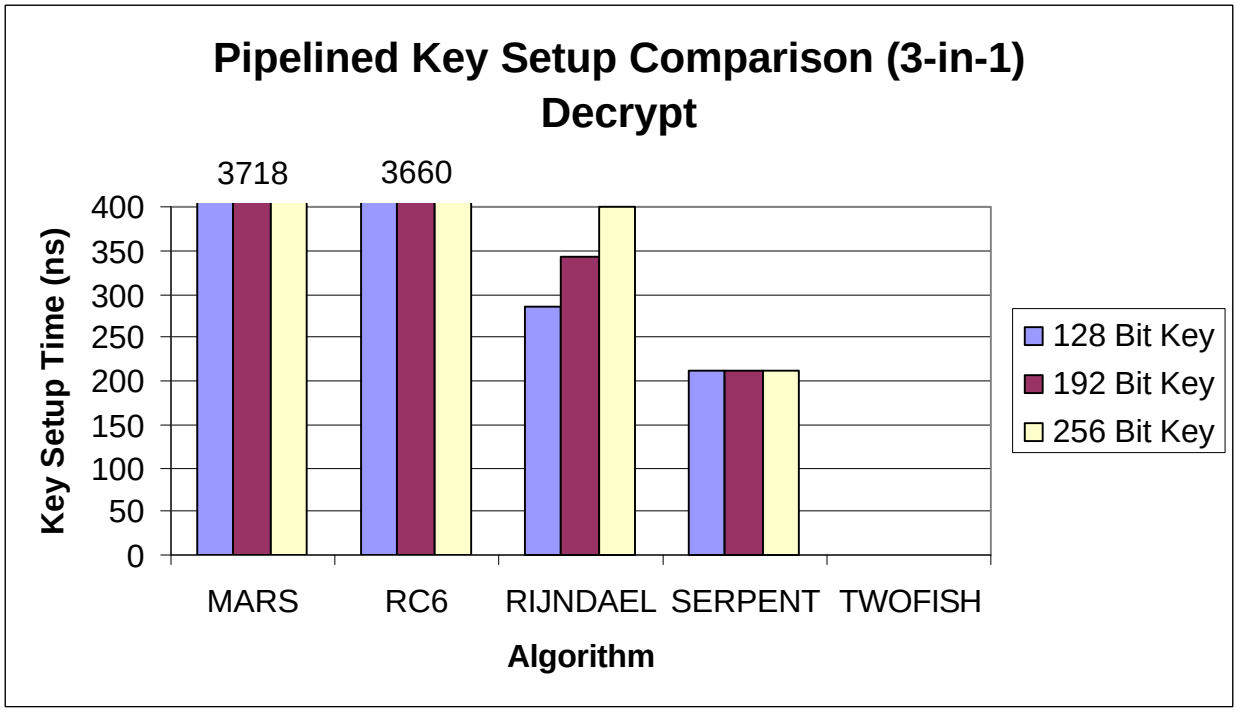

Figure 20 - Pipelined 3-in-1 Key Setup Comparison (Decrypt)

Figure 20 provides details on key setup times in the decrypt direction for the pipelined designs. TWOFISH provides the best performance in this area with no setup time in either encrypt or decrypt directions. As in the iterative case, the key setup time for RIJNDAEL and SERPENT increase in the decrypt direction relative to their corresponding encrypt direction setup times. Again, as in the iterative cases for SERPENT, the setup time in the decrypt direction can be reduced significantly, making SERPENT and TWOFISH comparable in the pipelined key setup cases. The times represent minimum times for all algorithms.

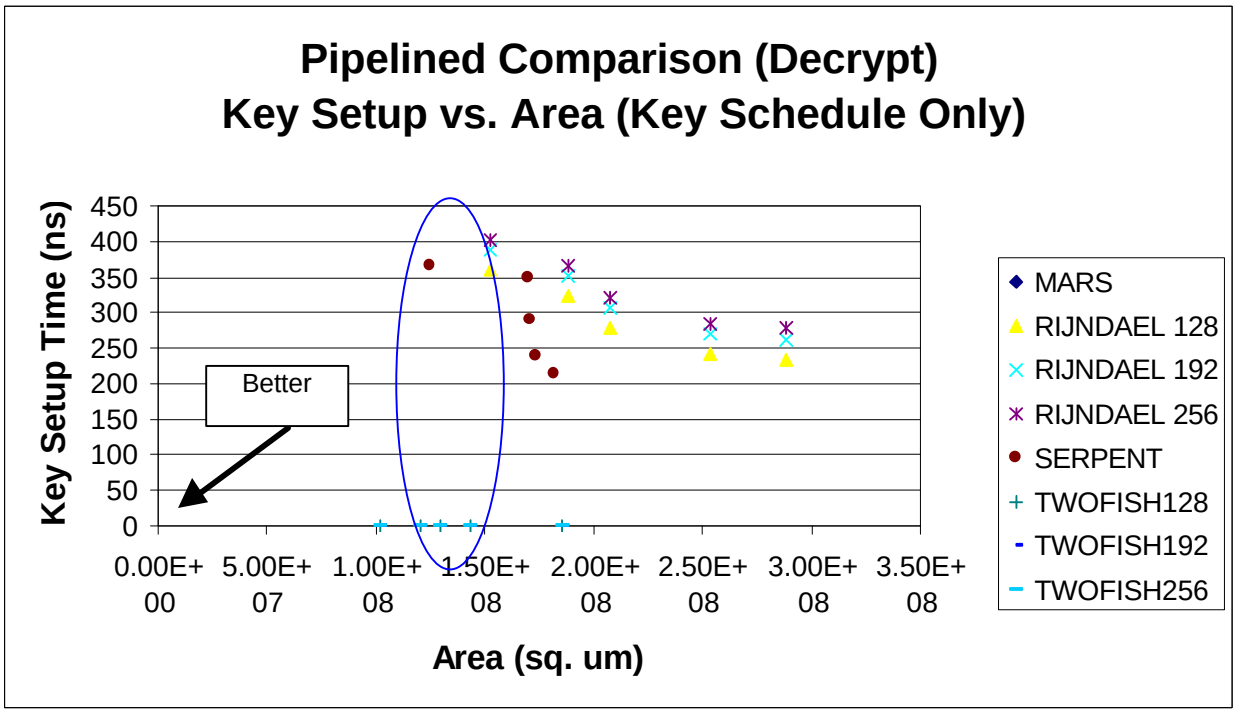

Figure 21 - Pipelined 3-in-1 Key Setup vs. Area (Decrypt)

Figure 21 summarizes the relationship between the amount of time required to generate subkeys for an algorithm and the amount of area required to implement the key setup. The algorithm portion is not included in this analysis. The optimal place for performance in this case is the lower left corner of the graph, with both key setup time and area minimized. The circled area highlights the performance variation for roughly the same area. In this case, TWOFISH provides the best level of performance for area in the decrypt direction. However, as stated previously, improvements to the SERPENT key scheduling in decrypt places it at almost the same level as TWOFISH (0 ns TWOFISH vs. 6.95 ns SERPENT). See section Key Setup Reduction for details on the SERPENT modification.

6.2.13 Pipelined Time to Encrypt

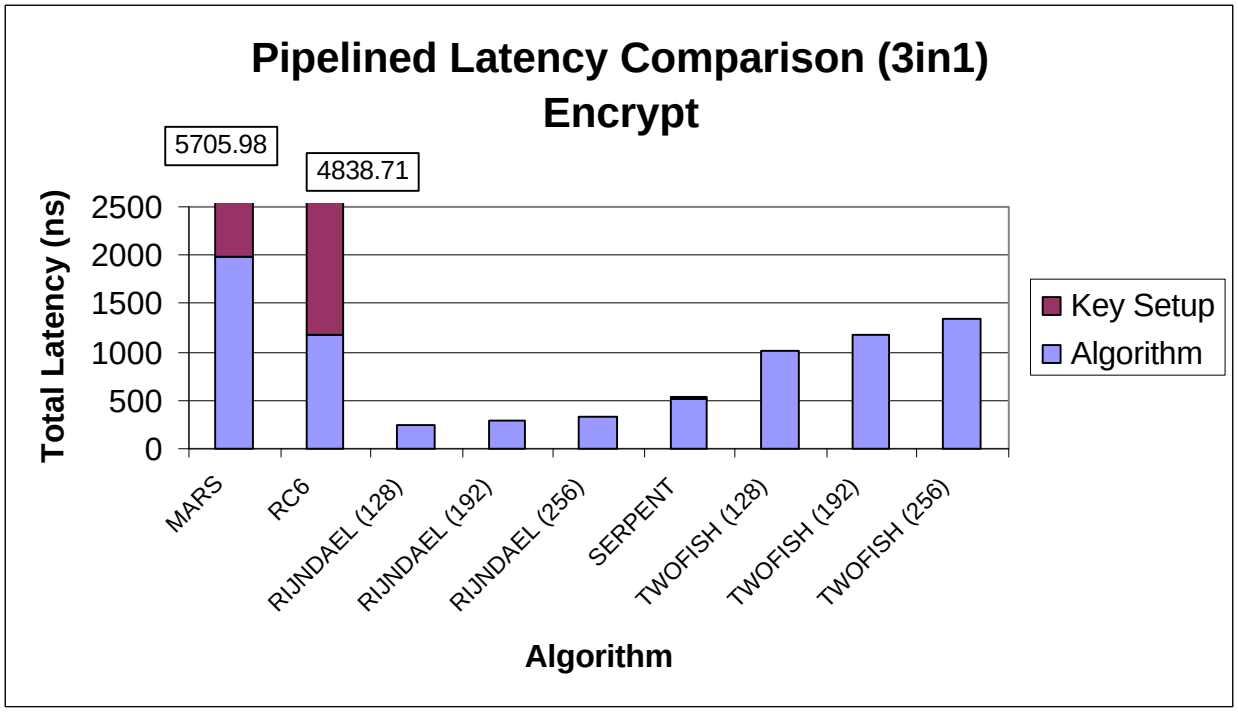

Figure 22 - Pipelined 3-in-1 Latency Comparison

Figure 22 describes the latency associated with propagating through the entire pipelined algorithms. Key setup and algorithm execution times are shown separately, but are most significantly notable in MARS and RC6 which are off scaled to 5706 ns and 4839 ns, respectively. (Note: Numbers above MARS and RC6 total times.) The minimum latency was realized by RIJNDAEL at 248 ns (128-bit key).

6.2.14 Pipelined Time to Decrypt

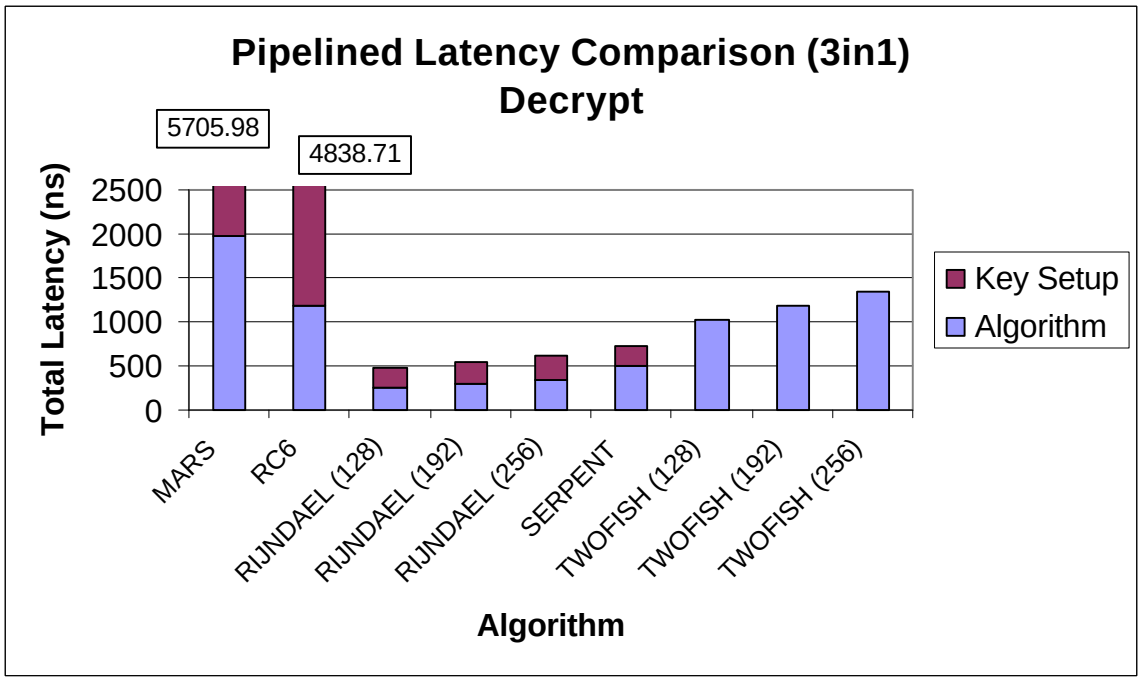

Figure 23 - Pipelined 3-in-1 Latency Comparison (Decrypt)

Figure 23 shows the total latency through each of the pipelined devices, including both key setup time and algorithm propagation times. RIJNDAEL provides the shortest total latency in total time, even with the additional setup time required in the decrypt direction. Key setup times for SERPENT and RIJNDAEL appear to be comparable, but the smaller number of rounds in RIJNDAEL reduces the total latency and provide better performance than SERPENT in this area. (check values for RC6 and MARS for accuracy, then insert comment as appropriate if, for example, the printed off scale number is for the key setup only vs. the total latency)

7 Performance Results – 128 Bit Key Design

7.1 Summary

A table summarizing the results and performance metrics is given below for algorithm comparison. These comparison values are given only for the 128-bit key size implementation. The effects of a single key size – especially a key size less than the maximum of the 3-in-1 case – are most notable in RIJNDAEL and TWOFISH since the number of rounds and internal round structure, respectively, are changed.

Parameter	Algorithm				
	MARS	RIJNDAEL	RC6	SERPENT	TWOFISH
Area (um2)	126,827,662	33,851,050	19,248,830	23,274,086	16,110,756
Transistor Count	1,941,371	641,681	307,247	345,483	264,058
Input/Outputs Required	520	520	520	520	520
Throughput (Mbps)	56.71	605.77	102.83	202.33	105.14
Key Setup Time Encrypt (ns)	9553	0	5742	19.77	60.87
Key Setup Time Decrypt (ns)	9553	211.3	5742	672.18	105.2
Algorithm Setup Time (ns)	0	0	0	0	0
Time to Encrypt One Block (ns)	2256.92	211.3	1244.8	632.64	1217.4
Time to Decrypt One Block(ns)	2256.92	211.3	1244.8	632.64	1217.4

Table 15 Iterated Summary

Parameter	Algorithm				
	MARS	RIJNDAEL	RC6	SERPENT	TWOFISH
Area (um2)	1,332,658,283	419,886,025	453,248,223	438,561,500	225,298,323
Transistor Count	20,661,116	4,292,749	7,536,366	5,741,469	3,783,973
Input/Outputs Required	520	520	520	520	520
Throughput (Mbps)	2189.16	5745.06	2196.67	8030.11	2273.53
Key Setup Time Encrypt (ns)	3712	0	3728.5	18.98	0
Key Setup Time Decrypt (ns)	3712	226.87	3728.5	212.55	0
Algorithm Setup Time (ns)	0	0	0	0	0
Time to Encrypt One Block (ns)	1987.98	222.8	1165.4	510	1126
Time to Decrypt One Block(ns)	1987.98	247	1165.4	510	1126

Table 16 Pipelined Summary

7.2 Performance Comparisons

7.2.1 Iterative Area

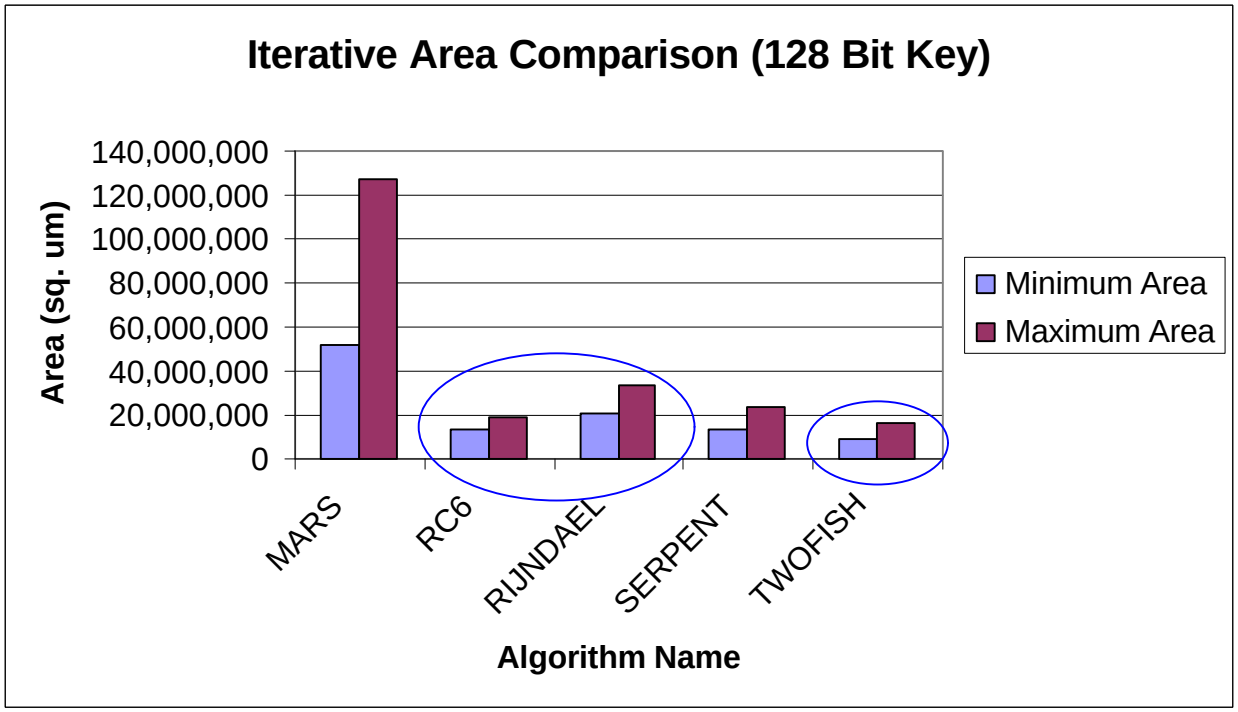

Figure 24 - Iterative 128 Bit Key Area Comparison

Figure 24 highlights the change in area for the 128-bit key iterative design. The circled area indicates the algorithms with significant area reductions compared with the 3-in-1 implementation. For the minimum area implementation of RC6, RIJNDAEL and TWOFISH, the area reduction for the 128-bit key only version is 10%, 14% and 28%, respectively. SERPENT and MARS are relatively unchanged.

7.2.2 Iterative Transistor Count

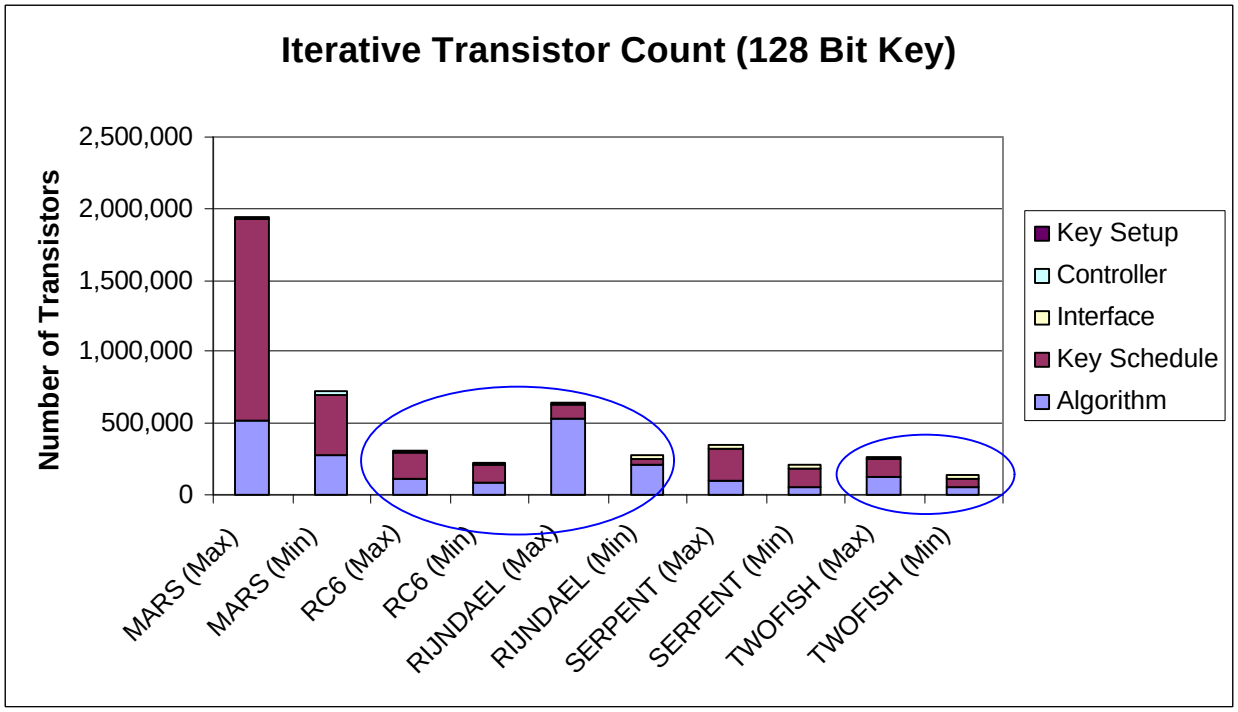

Figure 25 - Iterative 128 Bit Key Transistor Count Comparison

Figure 25 reflects the number of transistors required to implement the 128 bit key iterative design for all five algorithms. The totals include all of the major blocks in the iterative design, including the algorithm, key scheduling, interfacing, controller, and any expansion required. The minimum and maximum refer to the minimum and maximum area implementations. The circled area in the figure shows algorithms which exhibit a change in the transistor count between the 128-bit key and 3-in-1 implementations. The RC6, RIJNDAEL, and TWOFISH algorithms require fewer transistors to implement the 128-bit design by 13%, 43% and 26%, respectively, for the minimum area versions. SERPENT and MARS remain relatively unchanged.

7.2.3 Iterative Throughput

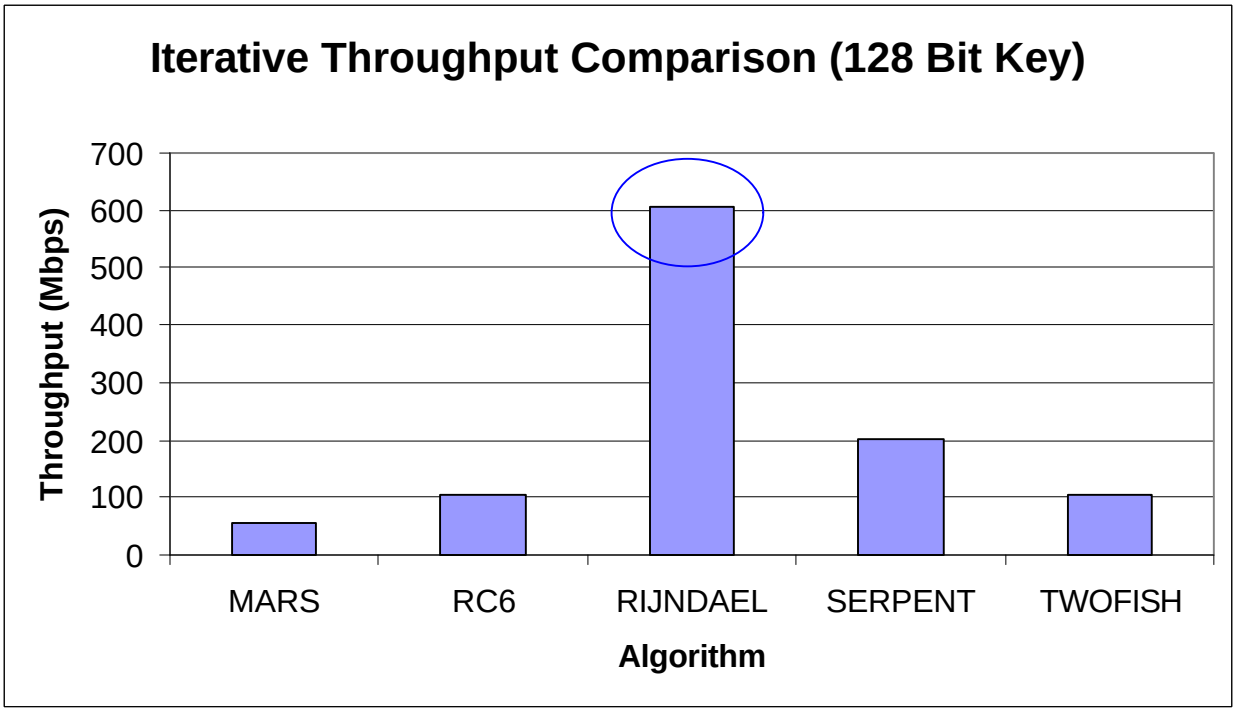

Figure 26 - Iterative 128 Bit Key Throughput Comparison

Figure 26 presents the iterative throughput values obtained for the 128 bit key size designs. The circled area highlights RIJNDAEL's increase in throughput from 450Mbps to 600Mbps. In this algorithm and design, the key schedule was the limiting factor in performance, so the reduction in the overhead and complexity necessary for a 3-in-1 design significantly boosted iterative throughput performance. The remaining algorithms were very similar to the 3-in-1 results.

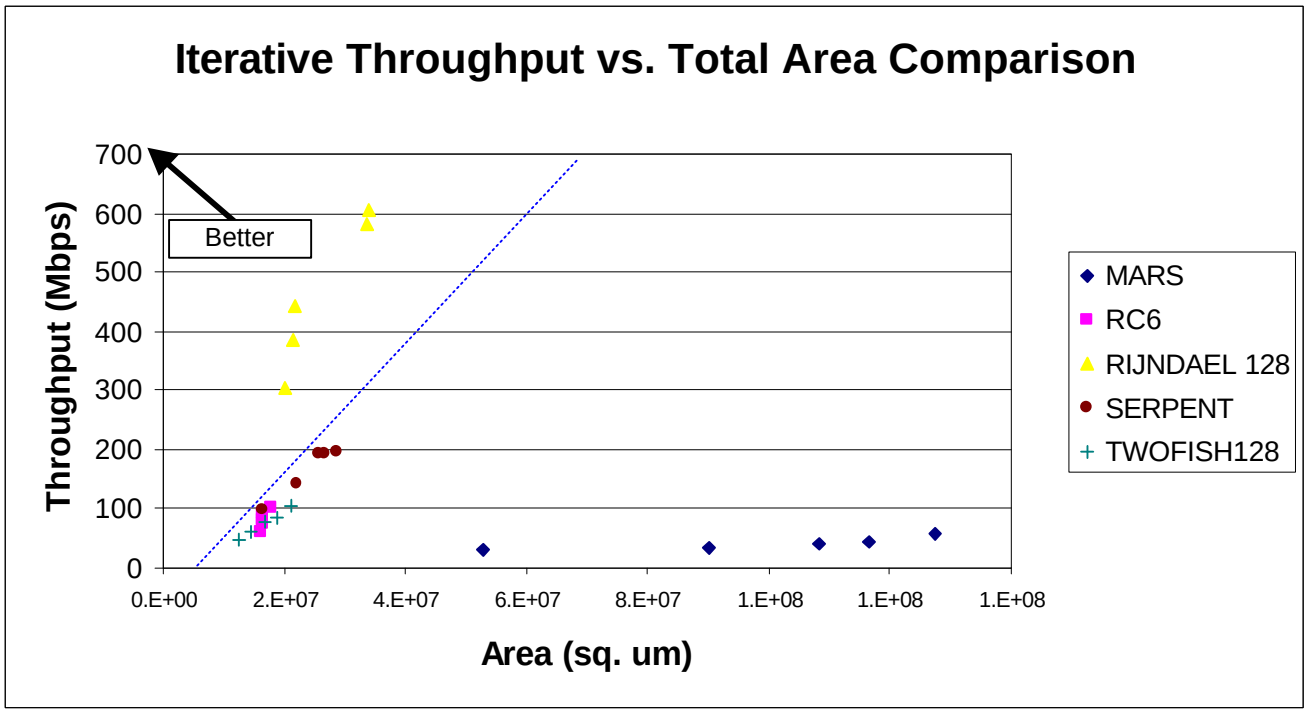

Figure 27 - Iterative 128 Bit Key Throughput vs. Area Comparison

Figure 27 shows the combined statistics of throughput and area for the iterative 128-bit key size version of the algorithms. The optimal location to be on the graph is in the upper left corner where throughput is maximized while area is minimized. As in the 3-in-1 case, RIJNDAEL shows the highest throughput overall, and achieves this at an area which is comparable to RC6, SERPENT and TWOFISH. As in the 3-in-1 case, MARS shows a very low throughput and a very large area.

To elaborate on the performance and area trade-offs, the following table summarizes the expected throughput per unit area.

Algorithm	Configuration					%Change (high-low)
	Speed 1	Speed 2	Speed 3	Speed 4	Speed 5	
MARS	0.446	0.379	0.365	0.385	0.554	52%
RC6	5.760	5.428	4.893	4.552	3.693	56%
RIJNDAEL	17.895	17.283	20.358	18.076	14.947	36%
SERPENT	6.804	7.151	7.525	6.482	5.892	28%
TWOFISH	4.978	4.517	4.510	4.086	3.857	29%

Table 17 - 128 Bit Key Throughput Per Unit Area (bps/sq. um)

In Table 18, each algorithm is represented for the 128 bit key size design. The configurations represent the varying speed grades of the design, where Speed 1 is the fastest, highest area design. Conversely, Speed5 is the slowest, least area intensive design. Each entry represents the amount of throughput (in bits/sec) that an algorithm will return given a 1 square micron unit of area. In this iterative case, RIJNDAEL still provides the highest throughput per area and for this case, the low end of RIJNDAEL is higher than the best throughputs for the remaining algorithms. The percent change column illustrates the increase is in throughput per area performance from the worst ratio (typically Speed 5) to the best ratio (typically Speed 1) for a given algorithm. It is interesting to note that MARS is the only algorithm that does not show an increasing ratio from Speed 5 to Speed 1. This characteristic of MARS makes it complicated and difficult to find an optimal hardware design.

7.2.4 Iterative Key Setup Time (Encrypt)

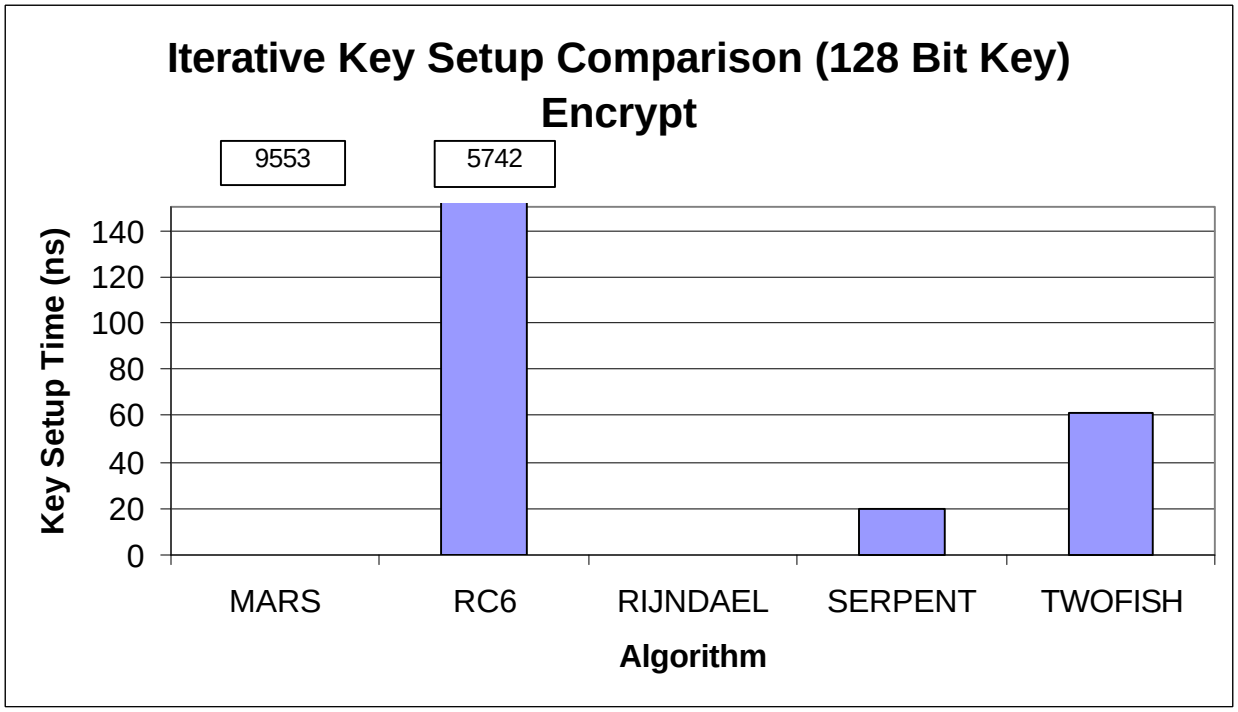

Figure 28 - Iterative 128 Bit Key Setup Comparison

Figure 28 provides a look at the iterative key setup for the 128-bit key design. In this case, the only significant change over the 3-in-1 case is the reduction of the key setup time for RC6. All other algorithms remain unchanged.

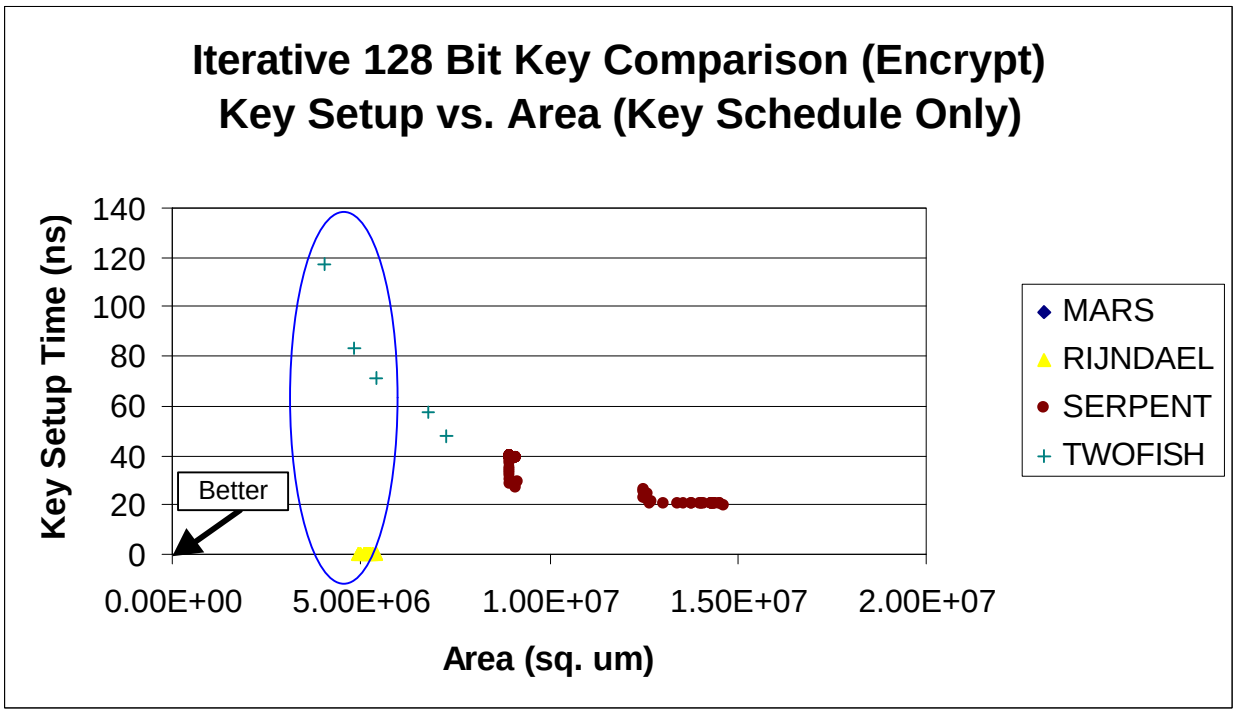

Figure 29 - Iterative 128 Bit Key Setup vs. Area Comparison

Figure 29 highlights the key setup versus area performance of the algorithms. Combining both parameters into a single graph, the optimal place for an algorithm to appear is in the lower left corner, with both key setup time and area minimized. Differences between this graph and the 3-in-1 design are primarily in the area reduction of SERPENT and TWOFISH. Both areas are reduced significantly, but as the circled area indicates, RIJNDAEL provides the best performance in this metric. MARS and RC6 are off-scale due to relatively long key setup times.

7.2.5 Iterative Key Setup Time (Decrypt)

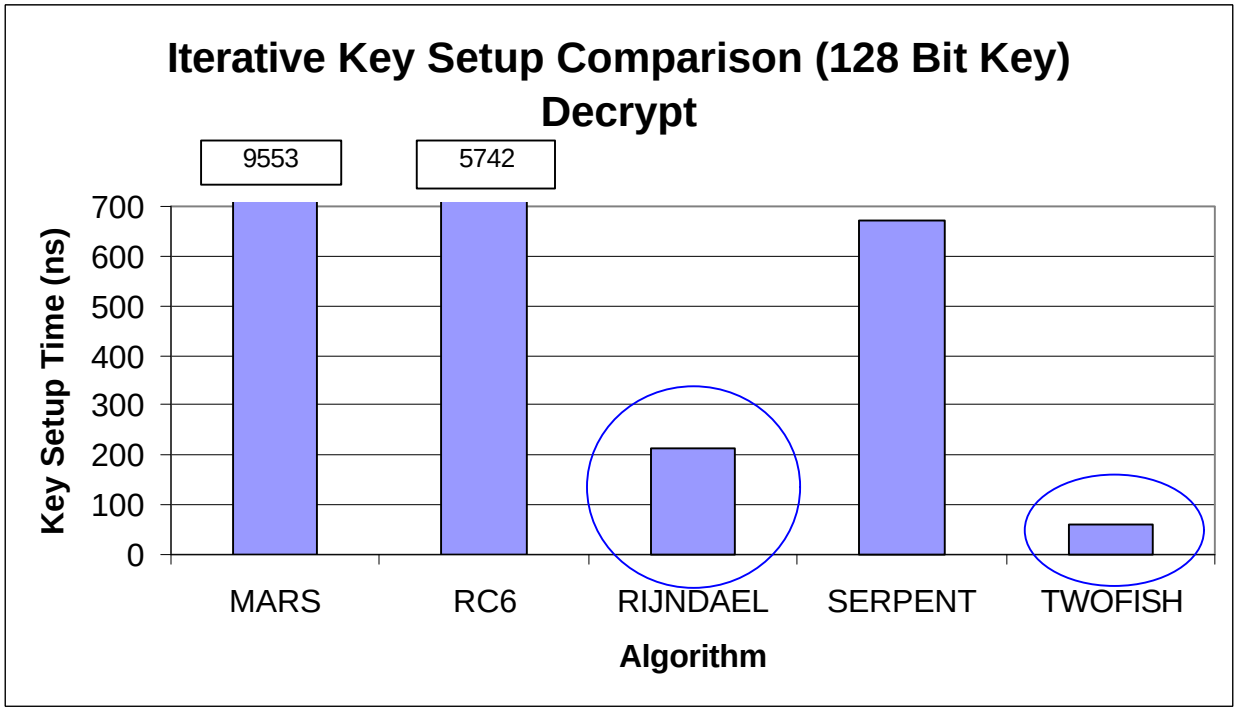

Figure 30 - Iterative 128 Bit Key Key Setup Comparison (Decrypt)

Figure 30 highlights the 128 bit key setup time for the decrypt direction. In this case, the RIJNDAEL times are reduced as compared to the 3-in-1 design. The TWOFISH key setup time is consistent with the TWOFISH128 statistic for the 3-in-1 case which outperforms the 192 and 256 bit performance. As in Figure 28, RC6 key setup times are also reduced.

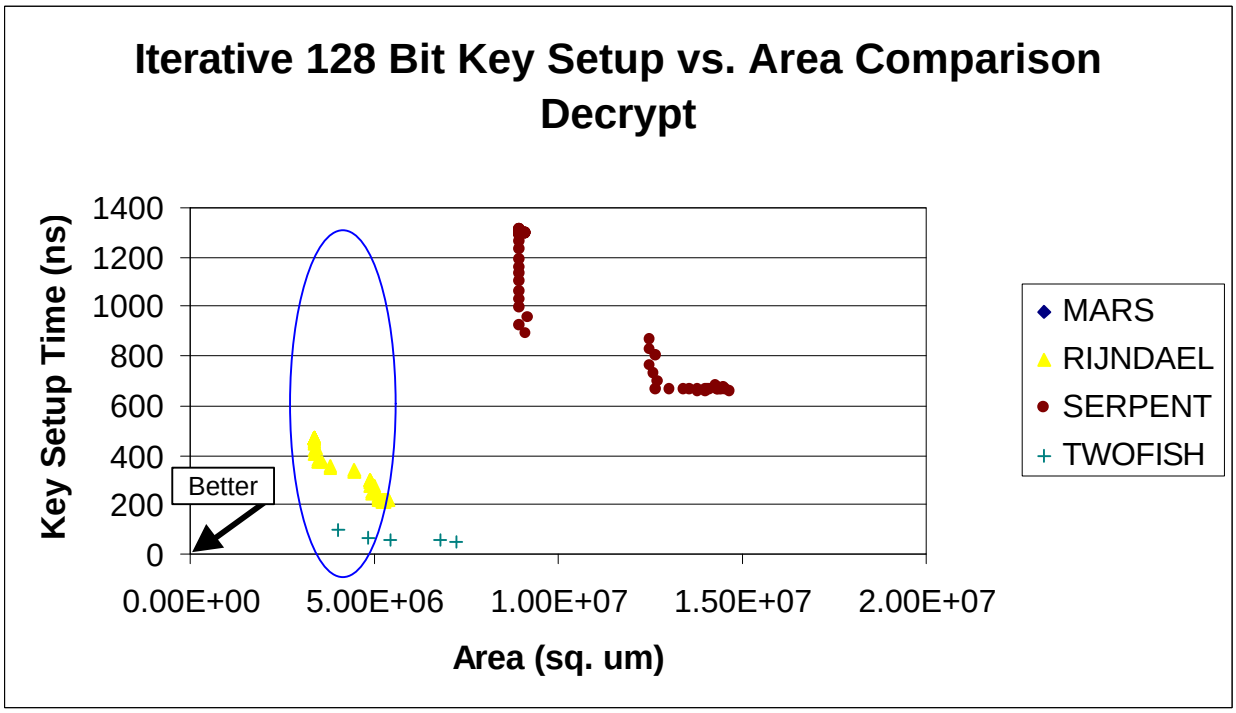

Figure 31 - Iterative 128 Bit Key Setup vs. Area Comparison (Decrypt)

Figure 31 examines the key setup versus area trade-off in the 128-bit key designs. From the encrypt direction, the positions of RIJNDAEL and TWOFISH have reversed, with TWOFISH providing the better performance for a given area.

7.2.6 Iterative Time to Encrypt

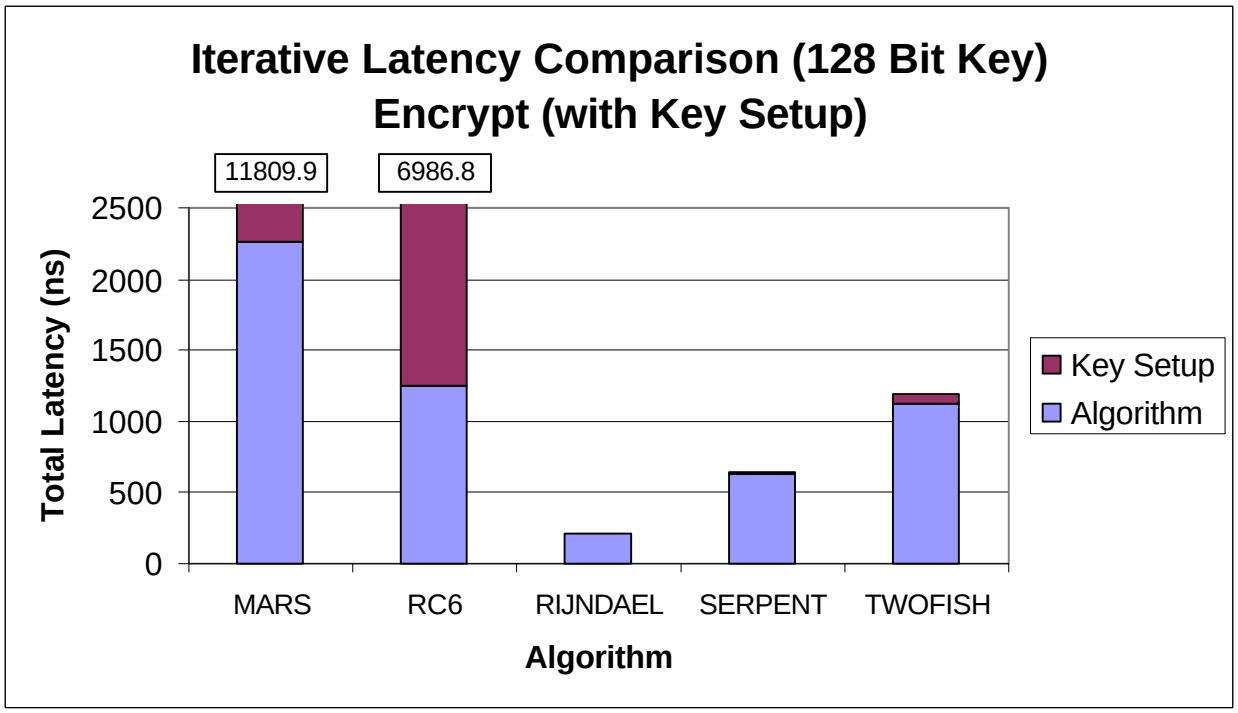

Figure 32 - Iterative 128 Bit Key Latency Comparison (Encrypt)

For the iterative 128-bit key case, RIJNDAEL sees the largest improvement in total latency for the encrypt direction over the 3-in-1 design. Improvements in the round time appear to improve performance significantly. MARS and RC6 go off-scale due to their relatively large key setup times.

7.2.7 Iterative Time to Decrypt

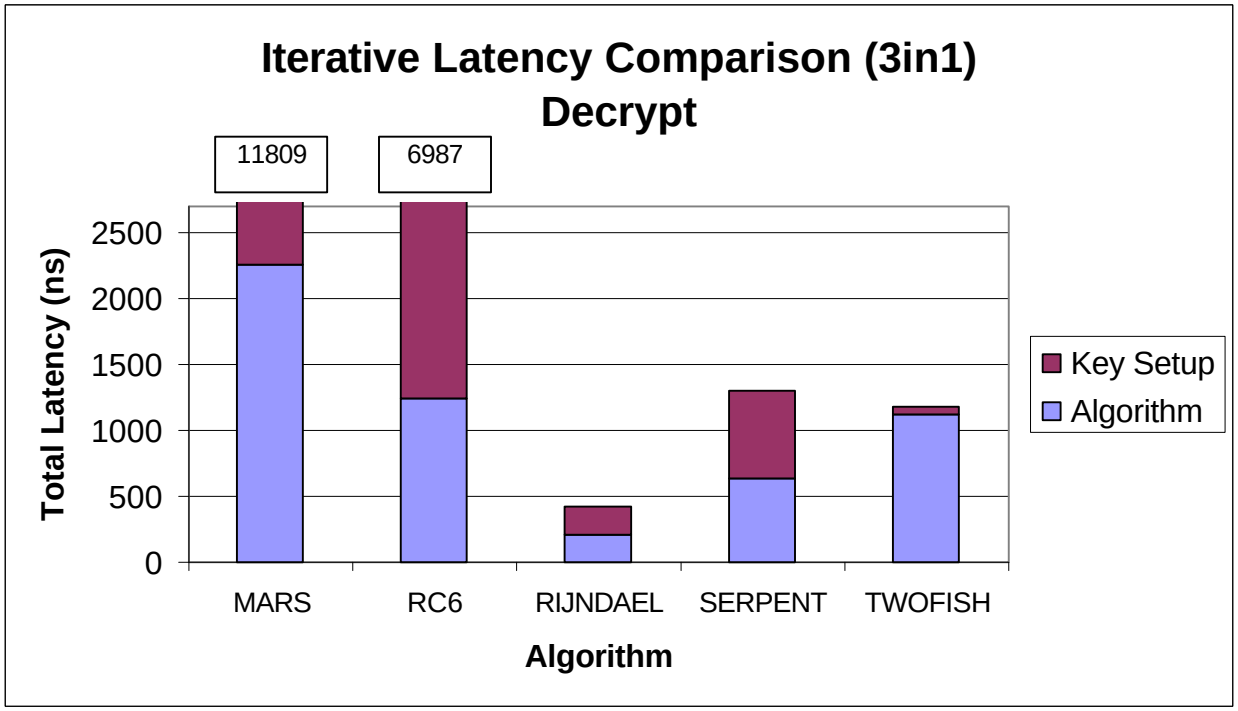

Figure 33 - Iterative 128 Bit Key Latency Comparison (Decrypt)

Figure 33 describes the relationship between the five algorithms for total decrypt latency. As in the previous graph, RIJNDAEL sees the largest improvement with the 128 bit key size and provides the lowest latency for decryption. MARS and RC6 go off-scale due to their relatively large key setup times.

7.2.8 Pipelined Area

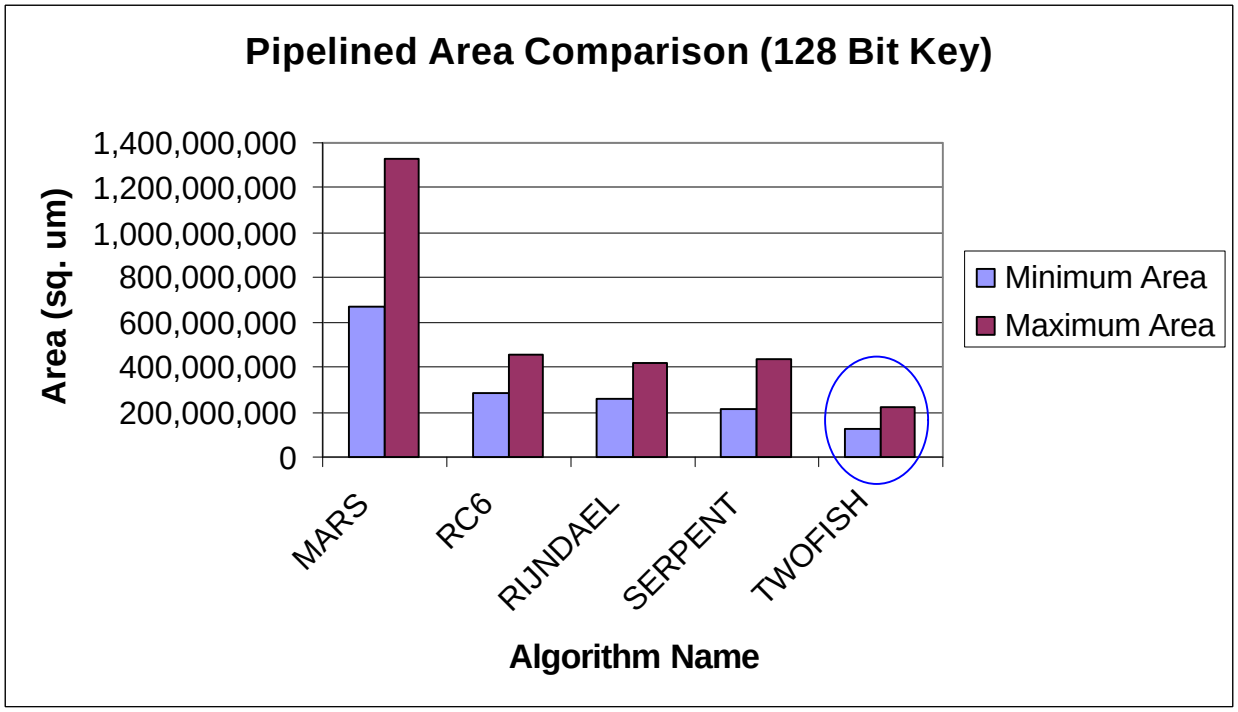

Figure 34 - Pipelined 128 Bit Key Area Comparison

Figure 34 shows the relationship of 128 bit key area across all five algorithms. In this case, the area required by TWOFISH is reduced dramatically over the corresponding 3-in-1 implementation, as shown in the graph. Also, RC6 and RIJNDAEL see reductions in total area.

7.2.9 Pipelined Transistor Count

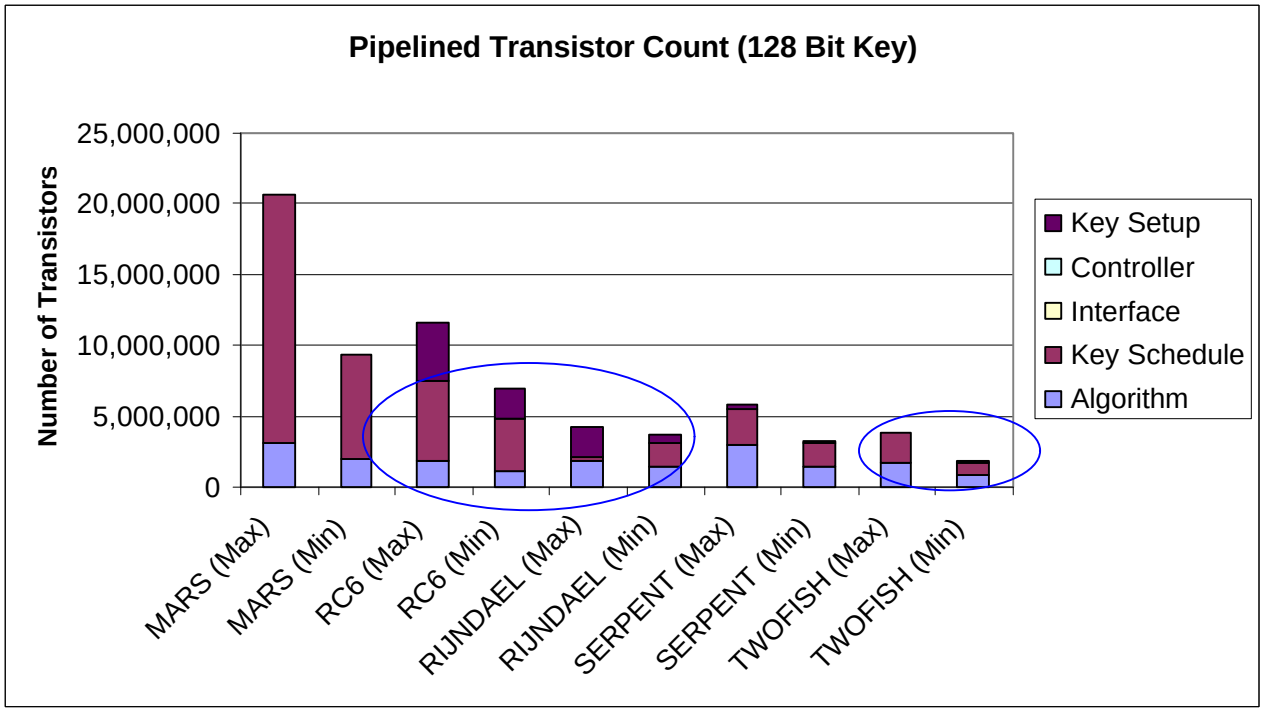

Figure 35 - Pipelined 128 Bit Key Transistor Count

As in the case of pipelined area, RC6, RIJNDAEL, and TWOFISH see significant reductions in transistor counts over the 3-in-1 designs while SERPENT and MARS remain the same size.

7.2.10 Pipelined Throughput

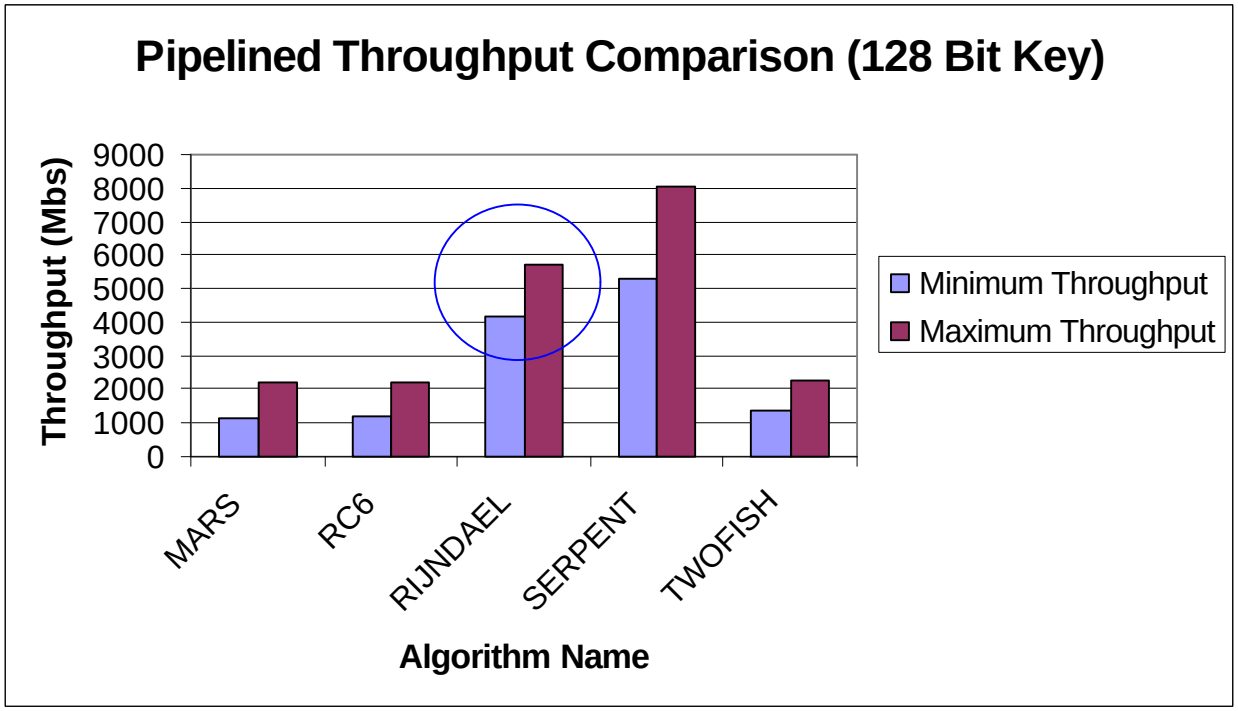

Figure 36 - Pipelined 128 Bit Key Throughput Comparison

An area to bring to light on Figure 36 is the increase in throughput of RIJNDAEL for 128 bit key sizes. Although SERPENT remains the highest throughput at 8 Gbps, RIJNDAEL is increased to approximately 6 Gbps in this design. The minimum throughput for TWOFISH is increased since the minimum throughput for the 3-in-1 case is the smallest throughput for the 256-bit key version. All other algorithms remain unchanged.

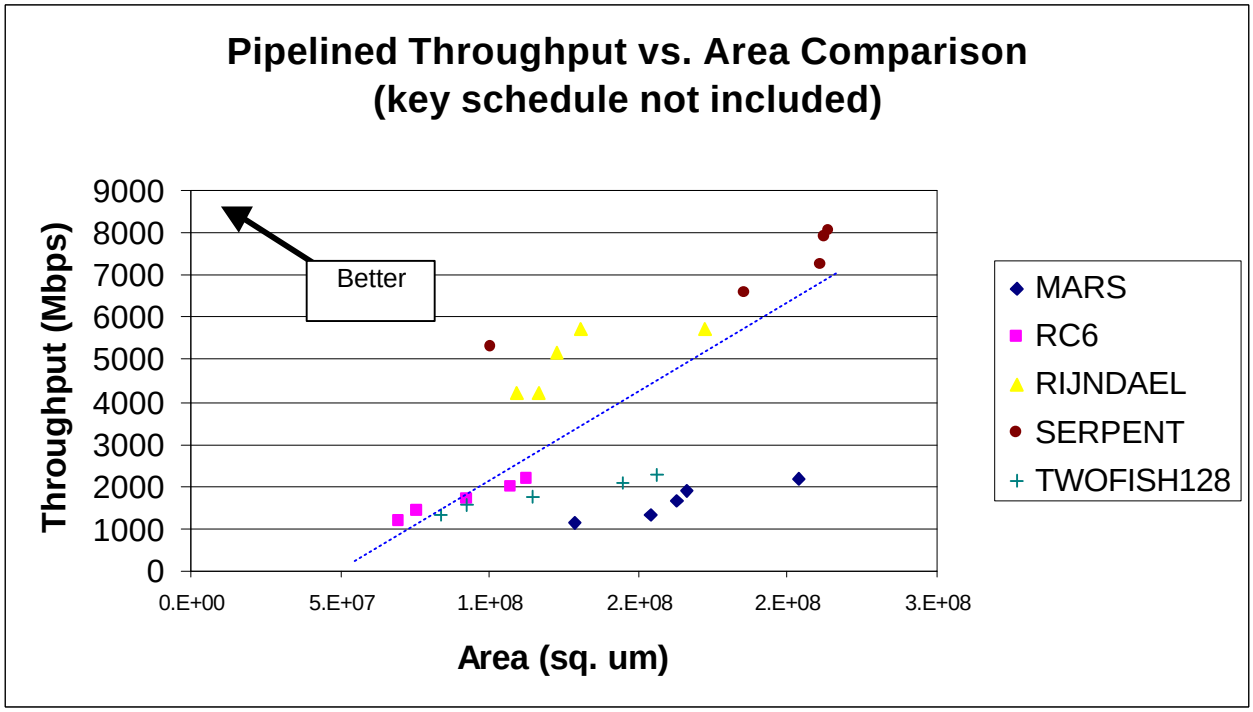

Figure 37 - Pipelined 128 Bit Key Throughput vs. Area Comparison

In Figure 40, the optimal location on the chart is the upper left where throughput is maximized while area is minimized. The only significant change for this metric relative to the 3-in-1 statistics is the movement of RIJNDAEL towards the upper left – an improvement. However, the best implementation of SERPENT still outperforms RIJNDAEL for this metric.

To elaborate on the performance and area trade-offs, the following table summarizes the expected throughput per unit area.

Algorithm	Configuration					%Change (high-low)
	Speed 1	Speed 2	Speed 3	Speed 4	Speed 5	
MARS	10.754	11.236	10.235	8.717	8.728	29%
RC6	19.289	18.550	18.363	18.632	17.053	13%
RIJNDAEL	33.371	43.988	42.239	36.060	38.529	32%
SERPENT	37.523	37.243	34.185	35.356	52.587	54%
TWOFISH	14.535	14.298	15.299	17.076	16.050	19%

Table 18 - 128 Bit Key Throughput Per Unit Area

In Table 18, each of the algorithms are represented for the 128 bit key size design The configurations represent the varying speed grades of the design, where speed 1 is the fastest, highest area design. Conversely, speed5 is the slowest, least area intensive design. Each entry represents the amount of throughput (in bits/sec) that an algorithm will return given a 1 square micron unit of area. In this pipelined case, SERPENT still provides the highest throughput per area and also the greatest increase in throughput from the low speed to high speed, with an increase of 54%. With reduced overhead in RIJNDAEL for the 128 bit key size, the maximum throughput of RIJNDAEL in bits per area starts to approach the high with SERPENT.

7.2.11 Pipelined Key Setup Time (Encrypt)

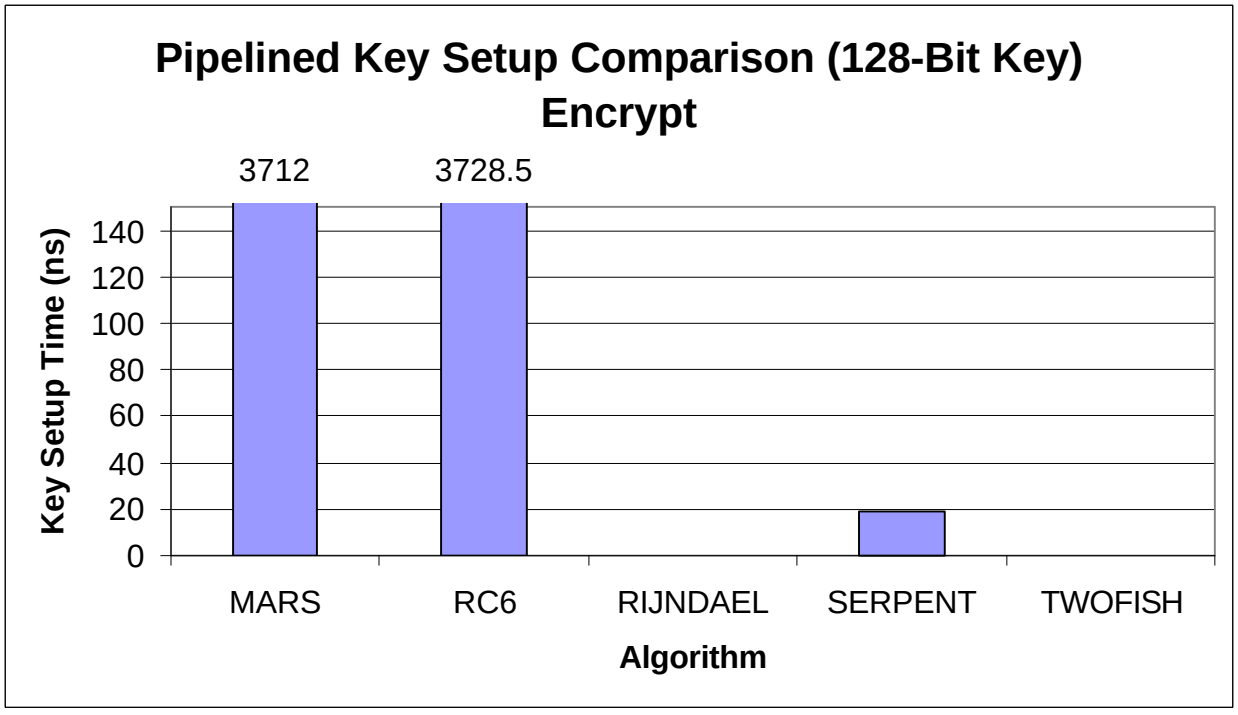

Figure 38 - Pipelined 128 Bit Key Key Setup Comparison (Encrypt)

There are no significant changes in key setup times for encryption.

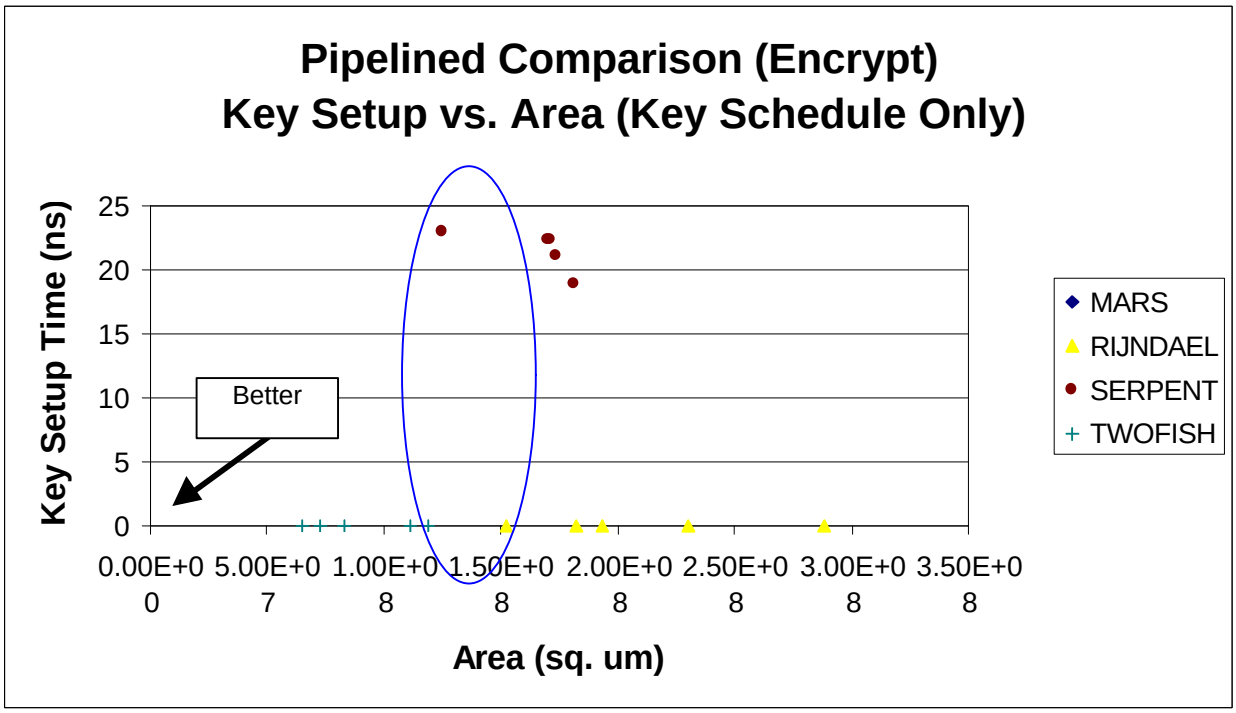

Figure 39 - Piplined 128-Bit Key Setup vs. Area Comparison (Encrypt)

Figure 39 shows the trade-off between key setup time and area for the encrypt direction. In this graph, the high end design of TWOFISH (circled) has been reduced from 185M (3-in-1 design) to 119M sq. um (128-bit design). The area performance of TWOFISH in this case has improved to where the largest design of TWOFISH falls at the smallest design of SERPENT, but still at a setup time of 0ns. RIJNDAEL's key setup time remains at 0 ns with approximately the same area as the 3-in-1 design, but as the circled region highlights, does not provide the same level of performance as TWOFISH. In the RIJNDAEL 128-bit design, the number of key scheduling functions that are used is equal to the worst case number of functions required for the 3-in-1 design. Consequently, there is no significant savings in area. However, the 192-bit and 256-bit do not require as many function blocks as compared to the 3-in-1 and 128-bit designs.

7.2.12 Pipelined Key Setup Time (Decrypt)

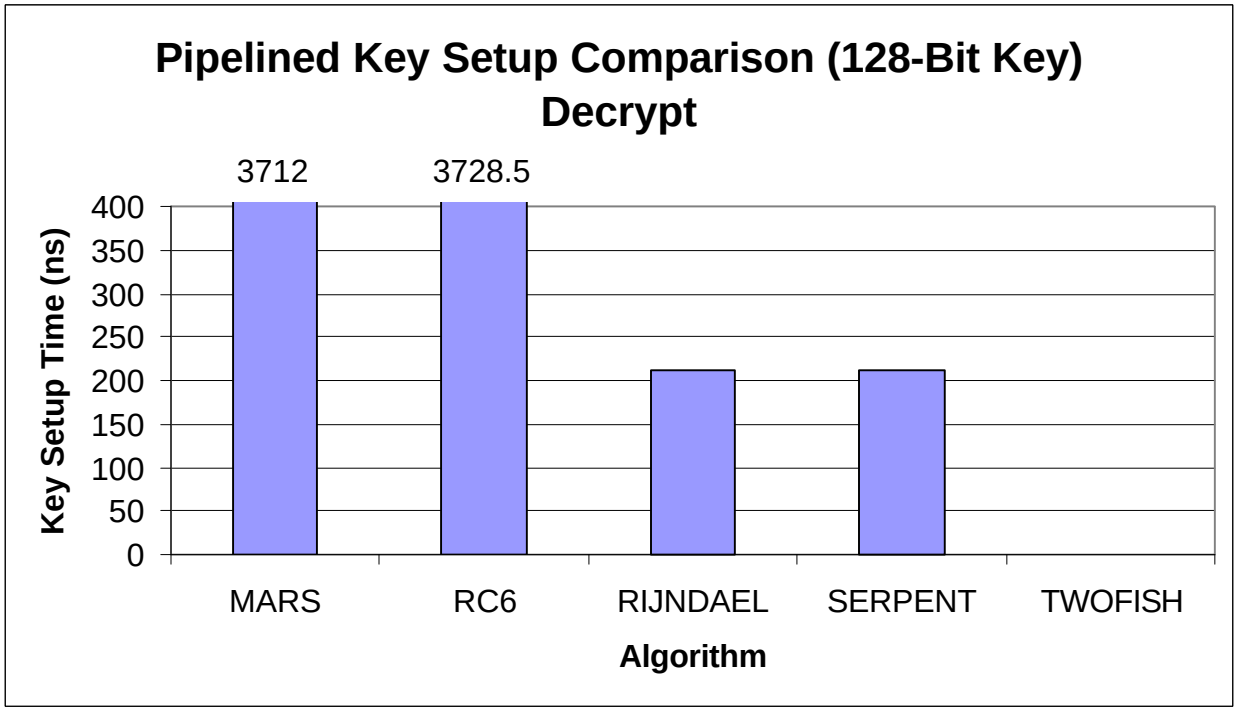

Figure 40 - Pipelined 128 Bit Key Setup Comparison (Decrypt)

In Figure 40, the most significant time reduction is with the RIJNDAEL algorithm. The remaining algorithms are unchanged.

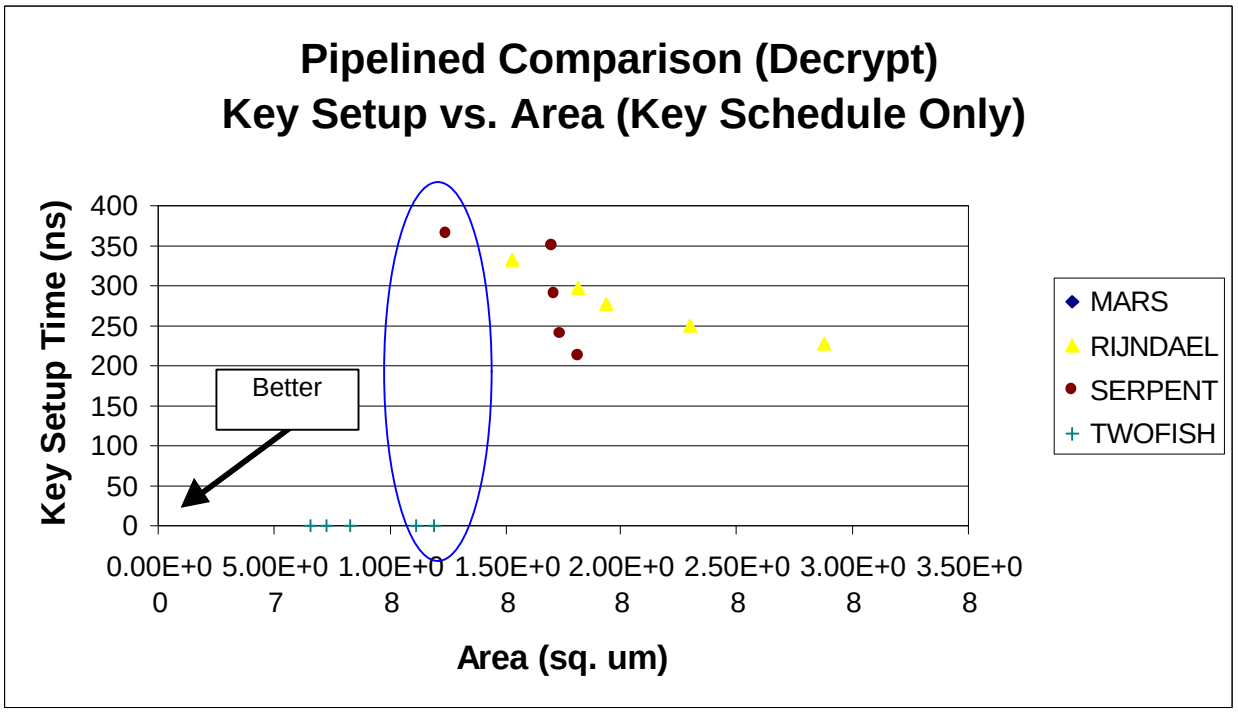

Figure 41 - Pipelined 128-Bit Key Setup vs. Area Comparison (Decrypt)

Figure 41 shows the trade-off between key setup time and area for the decrypt direction. In this graph, the high end design of TWOFISH (circled) has been reduced from 185M (3-in-1 design) to 119M sq. um (128-bit design). The area performance of TWOFISH in this case has improved to where the largest design of TWOFISH falls at the smallest design of SERPENT, but at a much improved setup time of 0ns. RIJNDAEL improves key setup time by about 4% with little or no change in area, as some of the multiplexing overhead is elminated. In the RIJNDAEL 128-bit design, the number of key scheduling functions that are used is equal to the worst case number of functions required for the 3-in-1 design. Consequently, there is no significant savings in area. However, the 192-bit and 256-bit do not require as many function blocks as compared to the 3-in-1 and 128-bit designs.

7.2.13 Pipelined Time to Encrypt

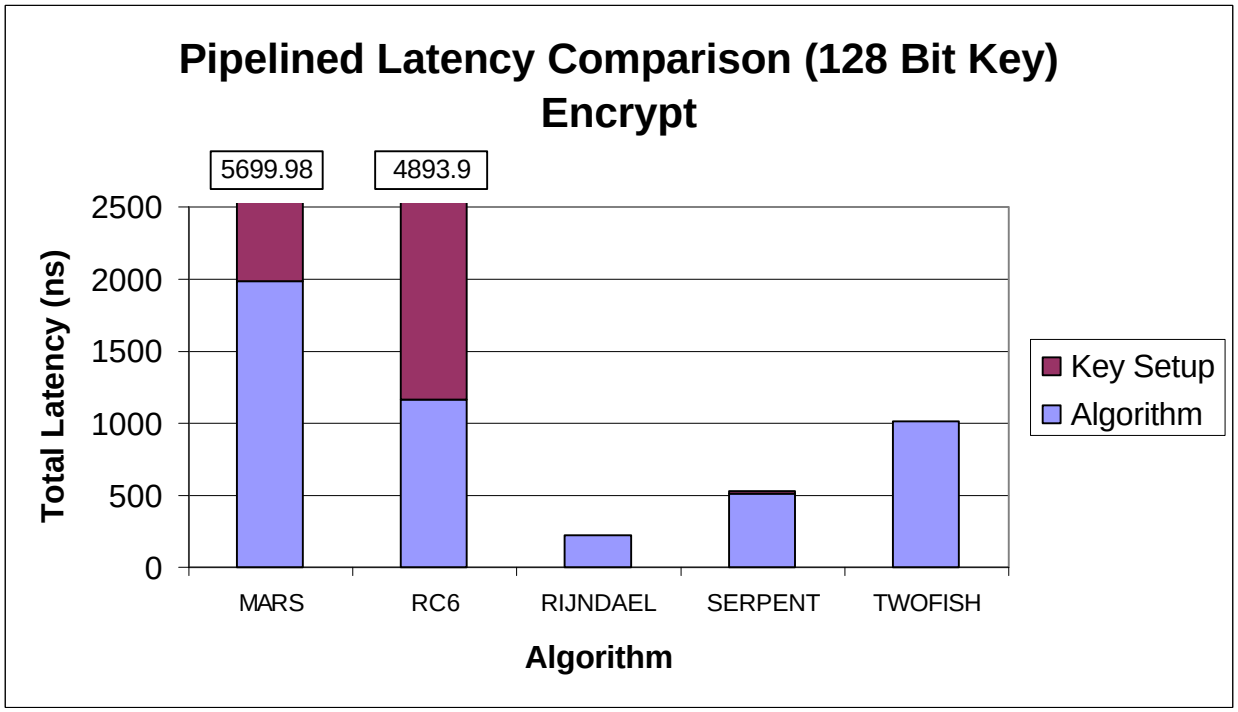

Figure 42 - Pipelined 128 Bit Key Latency Comparison (Encrypt)

Like the key setup time, the most significant change in the time required to encrypt a block when moving from the 3-in-1 design to the 128-bit key design is a sizeable improvement in the results for RIJNDAEL – approximately 10% improvement. MARS also improved by greater than 10%. However, its total latency remains off-scale and unusually high relative to RIJNDAEL, SERPENT and TWOFISH.

7.2.14 Pipelined Time to Decrypt

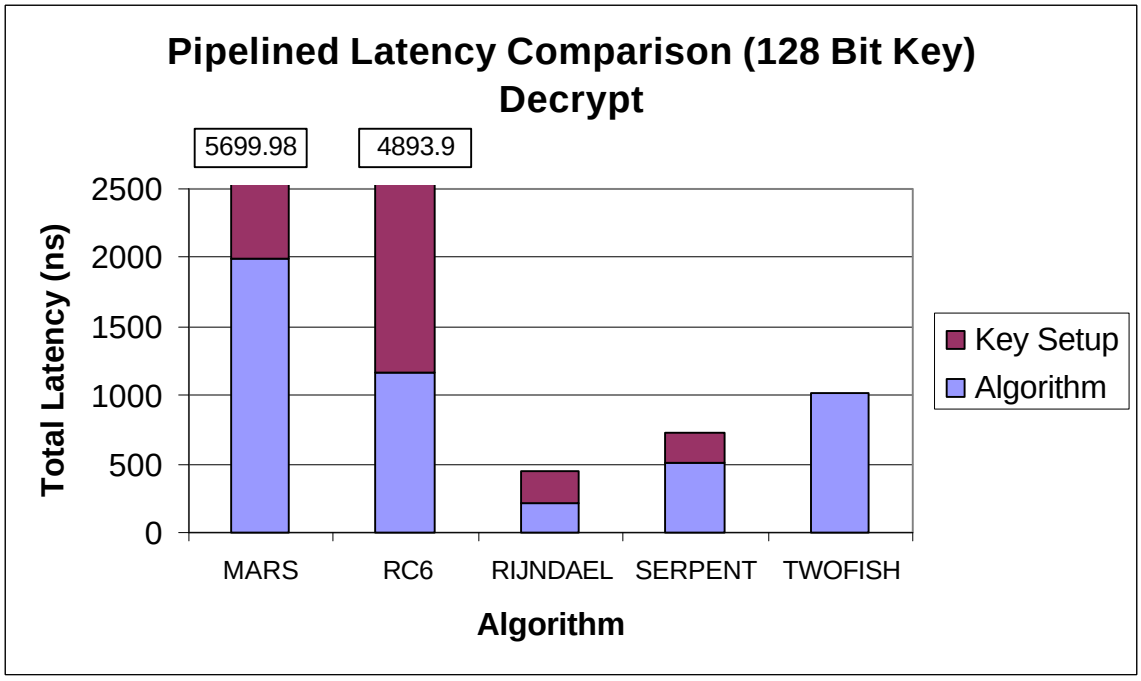

Figure 43 - Pipelined 128 Bit Key Latency Comparison (Decrypt)

With reductions in both algorithm and key setup for decryption, RIJNDAEL provides the lowest latency for all of the algorithms, with SERPENT following at approximately 700 ns.

8 Summary

This paper has presented an overview of the methods and architectures used for the AES hardware comparison. The primary characteristics used for design tradeoffs in hardware engineering are area and timing. As such, each algorithm was examined from the standpoint of minimum area (iterative architecture) and maximum throughput (pipelined architecture). Further, statistics and data based on area and timing were emphasized and illustrated for each algorithm.

The results (in Section 5.5.3) show vital parameters for both the iterative and pipelined architectures of each algorithm that can be used to evaluate relative performance. The designs were not optimized for any one parameter, but rather they serve as a good testbench scoring of all the algorithms relative to one another, given the same commonly used hardware design practices and procedures

A comparison was made between two varieties of implementations, namely a 3-in-1 where the key size is selectable to 128-bit, 192-bit or 256-bit, and a 128-bit only version. The results between the two implementations were not dramatic for MARS, RC6 or SERPENT. However, RIJNDAEL and TWOFISH showed some improvements worth noting for several metrics such as iterative throughput and overall area.

It should be emphasized that any data point based on a single parameter (e.g. transistor count or throughput) is a relatively narrow view of the algorithm's overall performance or rating. For example, an algorithm with the best transistor count and acceptable throughput may have a very long key setup time thus reducing key agility. For this reason, there was no attempt to rank algorithms in order. Rather, it is left to the cryptographic community to establish a consensus of the most important parameters – in combination or alone – and to draw appropriate conclusions from the data provided herein.

9 Acknowledgements

The authors would like to thank others at NSA for providing support to this project and in editing this report, namely Mr. Jeff Ingle. The authors also appreciate the opportunity NIST gave in encouraging this analysis, since they feel that high performance implementations for high-speed networks are an important aspect of the AES competition.

10 References

[1] W. Semancik, L. Mercer, T. Hoehn, G. Rowe, M. Smith-Luther, R. Agee, D. Fowlkes, and J. Ingle , "Cell Level Encryption for ATM Networks and Some Results from Initial Testing," , *DoD Fiber Optics Conference*, March 1994.

[2] C. Burwick, D. Coppersmith, E. D'Avignon, R. Gennaro, S. Halevi, C. Jutla, S. M. Matyas, L. O'Connor, M. Peyravian, D. Safford and N. Zunic, "Mars – a candidate cipher for AES," *First Advanced Encryption Standard (AES) Conference*, Ventura, CA, 1998.

[3] R. Rivest, M. Robshaw, R. Sidney, and Y. Yin, "The RC6™ Block Cipher," *First Advanced Encryption Standard (AES) Conference*, Ventura, CA, 1998.

[4] J. Daemen and V. Rijmen, "AES Proposal: Rijndael," *First Advanced Encryption Standard (AES) Conference*, Ventura, CA, 1998.

[5] R. Anderson, E. Biham, and L. Knudsen, "Serpent: A Proposal for the Advanced Encryption Standard," *First Advanced Encryption Standard (AES) Conference*, Ventura, CA, 1998.

[6] B. Schneier, J. Kelsey, D. Whiting, D. Wagner, and C. Hall, "Twofish: A 128-Bit Block Cipher," *First Advanced Encryption Standard (AES) Conference*, Ventura, CA, 1998.