Efficient High Throughput Low Area AES
Assistant Professor1-2, Student3-4-5-6
Department of ECE
Sri Shakthi Institute of Engineering and Technology, Coimbatore, India

Abstract:
Cryptography is the science of secret codes, enabling the secrecy and confidentiality of communication of data through channels. This protects against unauthorized parties. In general, a cryptographic system transforms the plain text into a cipher text, using most of the time a key. The Advanced Encryption Standard (AES), also known as Rijndael is an encryption standard used for secured transfer of information. AES is a block cycle algorithm that has been analyzed extensively and is now used widely. AES is a symmetric block cipher that is intended to replace DES. The Verilog code is synthesized on Spartan 3E FPGA using Xilinx Chip scope Pro software tool. The design was implemented on a Cyclone III (EP3C80F780C6 model) device. Both designs were synthesized using Quartus v9.1 tool. The achieved performance in the term of throughput of the proposed system/architecture is much higher than the other hardware implementations. The proposed system could be used for the implementation of integrity units, and in many other sensitive cryptographic applications, such as, digital signatures, message authentication codes and random number generators.

Keywords: Cryptography, AES, DES, FPGA, ADC, RAM, SOC.

I. INTRODUCTION
In today’s digital world, encryption is emerging as a disintegrable part of all communication networks and information processing systems, for protecting both stored and in transit data. Encryption is the transformation of plain data (known as plaintext) into unintelligible data (known as cipher text) through an algorithm referred to as cipher. There are numerous encryption algorithms that are now commonly used in computation, but the U.S. government has adopted the Advanced Encryption Standard (AES) to be used by Federal departments and agencies for protecting sensitive information. The National Institute of Standards and Technology (NIST) have published the specifications of this encryption standard in the Federal Information Processing Standards (FIPS). Any conventional symmetric cipher, such as AES, requires a single key for both encryption and decryption, which is independent of the plaintext and the cipher itself. It should be impractical to retrieve the plaintext solely based on the cipher text and the encryption algorithm, without knowing the encryption key. Thus, the secrecy of the encryption key is of high importance in symmetric ciphers such as AES. There are other important drawbacks in software implementation of any encryption algorithm, including lack of CPU instructions operating on very large operands, word size mis match on different operating systems and less parallelism in software. In addition, software implementation does not fulfill the required speed for time critical encryption applications. Thus, hardware implementation of encryption algorithms is an important alternative, since it provides ultimate secrecy of the encryption key, faster speed and more efficiency through higher levels of parallelism.

II. TESTING TECHNIQUES
Advanced mechanical systems and multimedia applications are increasingly required to provide hard real-time performance while maintaining low power. These requirements are leading to SoC solutions with multiple processor cores and active peripherals. Systems are further constrained by requirements to operate in harsh environments such as within a vehicle’s engine bay or even its gearbox. Development of embedded computer systems, where missed deadlines can result in physical damage to the mechanics, is an aggressive challenge. Efficient tools such as debuggers and profilers are an essential part of overcoming these challenges and are vital in the development of dependable embedded systems. Advancements in technology now permit the integration of an entire system onto a single silicon chip, known as a system-on-chip (SoC), which results in the existing external interfaces used extensively for development purposes being moved on-chip. Traditionally, the communications within an embedded system take place using the processor’s external system bus that is realised as tracks on a printed circuit board (PCB). The PCB supports convenient attachment points for the development tools that require a physical connection, such as logic analysers and storage oscilloscopes. The placement of the previously external interfaces on-chip leaves increasingly few points of attachment for external analysis tools, rendering developers ‘blind’ to the SoC internal state. Without a reliable and consistent view of an embedded system’s state, defects or bugs in the system can become difficult or even impossible to find. The primary solution to this lack of external interfaces is to make the internal nodes observable again from outside the system. There are many different approaches to achieving the required visibility, this review outlines them. Debug support can be defined as the strategy of placing access points within a system such as debuggers, profilers and calibration. Including both traditional bench equipment and basic on-chip debug support. Bench equipment includes logic analysers and storage oscilloscopes, triggered by internal detection units that observe the input signals or by on-chip debug support resources that signal events via a dedicated break output pin. Basic on-chip debug support provides facilities for remotely located development tools to run and control an embedded processor’s program via a development interface known as a ‘run-control’ interface. Basic on-chip debug support provides minimal interface to control the SoC. Software support can be active at runtime but will be in contention with the system tasks.
substantially impacting the system’s behaviour. Once halted, monitor routines provide a reliable means of observation and control. The process of stopping and starting the SoC, however, is not always practical, especially for real-time systems. Stopping and resuming usually changes the relationship between the system and the environment. When the SoC is stopped, the environment continues unless being simulated as in the case of a test bench. The intrusive nature of software instrumentation makes it unsuitable for hard real-time systems when used with the ‘real’ production mechanics. Any instrumentation included during debugging would be removed in the product, thereby changing the overall behaviour. A second disadvantage of this strategy is that it only supports ‘post-mortem’ debugging that provides observation once the errors are visible, not when they actually occur. Software instrumentation is, however, very useful in some applications and is much safer to use in systems with soft deadlines, such as those not controlling the actual mechanics, particularly if the system is not operating at maximum capacity. Furthermore, a recent study has shown that for a hard real-time system designed to withstand the non-determinism of using cached memory, carefully optimised light weight software instrumentation only slightly reduces determinism when profiling high-level tasks.

III. PROPOSED SYSTEM

ON CHIP DEBUGGER CORES

The On chip debugger program consists of two basic debugging tools. These are Core Generator and Core Inserter. The Core Generator tool is a graphical user interface used to generate different type of cores inside the FPGA. On the other hand, Core Inserter is a post-synthesis tool used to debug functionality and analyze an already synthesized design without any HDL instantiation.

The AES algorithm is currently the standard

Finally in October 2000, the Rijndael algorithm was chosen as the basis for the new standard encryption algorithm (Hironobu 2001). The original Rijndael algorithm also supported both fixed-size and variable-size bit cipher blocks. However, currently the Federal Information Processing Standards specification for the AES algorithm supports only the fixed-size, 128-bit blocks. The operation of the AES algorithm is shown. The encryption step uses a key that converts the data into an unreadable cipher text, and then the decryption step uses the same key to convert the cipher text back into the original data. This type of key is a symmetric key; other algorithms require a different key for encryption and decryption. The precise steps involved in the algorithm can be seen. The process is relatively simple, but some brief cryptographic explanations are necessary to understand what is going on. In cryptography, algorithms such as AES are called product ciphers. For this class of ciphers, encryption is done in rounds, where each round’s processing is accomplished using the same logic. Moreover, many of these product ciphers, including AES, change the cipher key at each round. Each of these round keys is determined by a key schedule, which is generated from the cipher key given by the user.

![Figure 2. AES system](http://ijesc.org/)

IV. THROUGHPUT CALCULATION

The Avalon® interface communicates a 32-Bit DWORD per clock cycle. Therefore a key is transmitted in 4 to 8 cycles plus one cycle to activate key expansion with the control word 3.1. A payload data block or the result consist always of 4 DWORDs, thus it takes 4 cycles to send data to the core, one cycle to activate the computation with the control register 3.1 and 4 cycles to retrieve the data. The key expansion component computes one column of a round key in two clock cycles. In the first cycle the column is substituted through the s-box, in the second cycle the shift-operation is executed. AES specifies [1], depending on the key length \( N_{\text{rounds}} \) is \( \{10, 12, 14\} \) round keys with 4 columns each. The FSM of the key expansion module adds a clock cycle for the “DONE” state.

\[
T_{\text{key expansion}} (N \text{ round keys}) = 2 \cdot 4 \cdot N \text{ round keys} + 1
\]

The key expansion therefore takes 81, 97 or 115 clock cycles until the encryption or decryption can start. The round keys are stored until invalidated, see 4 thus this step is only needed once after power-up until the key changes. The AES core computes one iteration (round) of the Rijndael-Algorithm each clock cycle, thus a 128 Bit data block is encrypted or decrypted in 10, 12 or 14 cycles plus an initial round. The maximum throughput \( T_{\text{max}} \) [Bits] depends on the maximum operation frequency \( f_{\text{max}} \) and the key length which influences the number of rounds \( N_{\text{max}} \).
\[ T_{\text{max}} = (1 + N^{\text{rnd}}) \cdot 128 \text{Bit} \] (2) \[ f_{\text{max}} \]

Note: Equation 2 assumes that the round keys are already generated and does not include the constant of 4+1+4 Avalon\textsuperscript{®} bus cycles for transmission of data, activation and result retrieval.

V. OPERATION AND FUNCTIONALITY OF THE SYSTEM

Protocol Sequence

The AES component appears as memory mapped peripheral. All writes are fundamental slave write transfers, see [4] and take one clock cycle of the Avalon\textsuperscript{®} bus clock clk. It is not necessary to write all words of a input parameter successively or in one transfer. Bursts are not supported. Before any AES operation can start the initial user key has to be written to KEY segment of the memory map. After the user key is transferred to the component the KEY_VALID Bit must be set to start the key expansion. This Bit can be set simultaneously with DEC or ENC Bit of the control register. To invalidate the previous key and use another key the KEY_VALID must be deasserted for at least one Avalon\textsuperscript{®} bus clock cycle during this cycle the new key can already be transferred. Once a key is passed and marked valid data blocks can be transferred to the DATA segment of the memory map. The AES operation is started by asserting the ENC Bit for encryption or DEC Bit for decryption. While asserting ENC or DEC the KEY_VALID Bit must be kept asserted. The ENC or DEC Bit respectively deasserted by the component after completing the requested operation. The result of the operation can be read from the RESULT area of the memory and is not cleared. It will be overwritten by succeeding operations. NO_ROUNDS is the total number of rounds the processing takes, a constant defined by the generic KEYLENGTH 2.1. The AES standard in[1] defines 10 rounds for 128 Bit key, 12 rounds for a 192 Bit key and 14 rounds for a 256 Bit key. Thus depending on the key length the processing of a data block needs at maximum 15 clock cycles from data_stable=1 to completion, if the key is already expanded.

Interrupt Behavior

By setting IRQ_ENA in the control register 3.1 the component is configured to issue interrupt requests. If IRQ_ENA is asserted the interrupt request IRQ 2.2 will be set when the computation has completed in addition to clearing the ENC or DEC Bit. The IRQ 2.2 signal will remain set until clearing IRQ_ENA or a read operation on the RESULT area of the components address range.

THE INNER CORE

The algorithmic core is divided into two separate data paths one for encryption and a second for decryption operation. The two data paths are independent, however they share the key expansion component which provides decrypt and encrypt keys (which are the same only in opposite order). Each data path is controlled by its own FSM. If configured by the generic DECRYPTION 2.1 the decryption data path is included and some multiplexers are generated for the shared signals, e.g. result or roundkey_index. For reference the encryption data path of aes_core.vhd is given in figure 4.1.

VI. OUTPUT

The Analysis of the Output waveform for proposed design is given below with its simulated output. The simulated output in the XILINX ISE with its extended output are shown below the below figure shows AES encryption chipper text in binary form of 128 bit, using modelsim. The proposed AES system is simulated with the input clock, state and key, where state and key is 128 bit

![Figure 4. AES 128 bit waveform.](image)

![Figure 5. Individual Bit in Cipher Text 0 bit to 19 bit](image)

VI. CONCLUSION

Optimized and Synthesizable Verilog code is developed for the implementation of both encryption processes. Each program is
tested with some of the sample vectors and output results are perfect with minimal delay. Therefore, AES can indeed be implemented with reasonable efficiency on an FPGA; with the encryption taking an average of 320 and 340 ns respectively (forevery128bits).Thetimevariesfromchiptochipandthecalculate
delay time can only be regarded as approximate. Adding data pipelines and some parallel combinational logic in the key scheduler and round calculator can further optimize this design.

VII. REFERENCE


