IMPLEMENTACIÓN DEL ALGORITMO DE CIFRADO AES EN UN PROCESADOR NIOS II

Cyntia E. Enríquez Ortiz, Raúl Fernández Zavala
Centro de Innovación y Desarrollo Tecnológico en Cómputo (CIDETEC)
Unidad Profesional López Mateos, Instituto Politécnico Nacional

Resumen

En este trabajo se describe la arquitectura en un FPGA para la implementación de instrucciones específicas de cifrado/descifrado AES en un procesador NIOS II, como una alternativa a la implementación del algoritmo basada únicamente en software o en circuitos integrados dedicados especialmente al cifrado. Esta alternativa combina la flexibilidad de la solución en software con la eficiencia y desempeño de la solución de hardware dedicado.

Palabras Clave: AES, cifrado, FPGA, NIOS.

I. Introducción

Los algoritmos de cifrado se clasifican básicamente en dos tipos: cifrado simétrico o de clave privada y cifrado asimétrico o de clave pública. En el cifrado simétrico se utiliza la misma clave para cifrar y descifrar mensajes, y entre los algoritmos más utilizados se encuentra el algoritmo AES (*Advanced Encryption Standard*) [**?**]. Desde que el algoritmo AES fue adoptado como un estándar de cifrado en el año 2001 por parte del gobierno de los Estados Unidos, se han reportado diversos trabajos orientados a la implementación de procesadores o arquitecturas de aplicación específica en cifrado AES. Estos trabajos cubren diferentes estructuras o enfoques: arquitecturas compactas que optimizan el uso de recursos disponibles en un FPGA [**?**], diseño de ASIPs de 8 bits para cifrado AES [**?**], procesadores para cifrado basados en arquitecturas disparadas por transporte (TTA, *Transport Triggered Architecture*) [**?**], arquitecturas RISC de 32 bits de propósito especial optimizadas para la ejecución de algoritmos de criptografía mediante bloques coprocesadores [**?**], y procesadores con arquitectura VLIW (*Very Long Instruction Word*) para cifrado simétrico [**?**, **?**, **?**]. Otros enfoques adoptados recientemente para acelerar el cifrado de datos son la utilización de las Unidades de Procesamiento Gráfico [**?**, **?**] y la extensión del conjunto de instrucciones de procesadores de propósito general [**?**, **?**].

II. El algoritmo AES

El algoritmo AES [**?**] es un algoritmo de cifrado simétrico de bloques, en el cual el transmisor y el receptor utilizan una sola llave tanto para el cifrado como el descifrado. El tamaño del bloque de datos es de 128 bits, mientras que la longitud de la llave puede ser de 128, 192 o 256 bits. El algoritmo es iterativo y cada iteración se denomina ronda; el número de rondas puede ser 10, 12 o 14 dependiendo de la longitud de la llave. Internamente, el bloque de datos de 128 bits se representa como un arreglo bidimensional de bytes llamado *estado*, y todas las operaciones del algoritmo AES se realizan sobre este arreglo. Cada byte en el arreglo de estado se denota por S_{rc} y es considerado como un elemento de un campo finito $GF(2^8)$. El polinomio irreducible utilizado en AES es:

$$m(x) = x^8 + x^4 + x^3 + x + 1 \tag{1}$$

En cada ronda del cifrado, excepto la final, se realizan cuatro transformaciones: la sustitución de bytes `SubBytes()`, el desplazamiento de filas `ShiftRows()`, la mezcla de columnas `MixColumns()`, y la suma de llave de ronda `AddRoundKey()`. La ronda final no incluye la transformación `MixColumns()`. Para el descifrado se utiliza la transformación inversa correspondiente a cada transformación utilizada en el cifrado: la sustitución de byte inversa se denota `InvSubByte()`, el desplazamiento de filas inverso se denomina `InvShiftRow()`, la

mezcla de columnas inversa se indica por `InvMixColumns()` y finalmente la transformación inversa para la suma de llave de ronda es la misma transformación `AddRoundKey()`.

En la Figura 1 se muestra el diagrama a bloques de la estructura correspondiente al cifrado y descifrado AES.

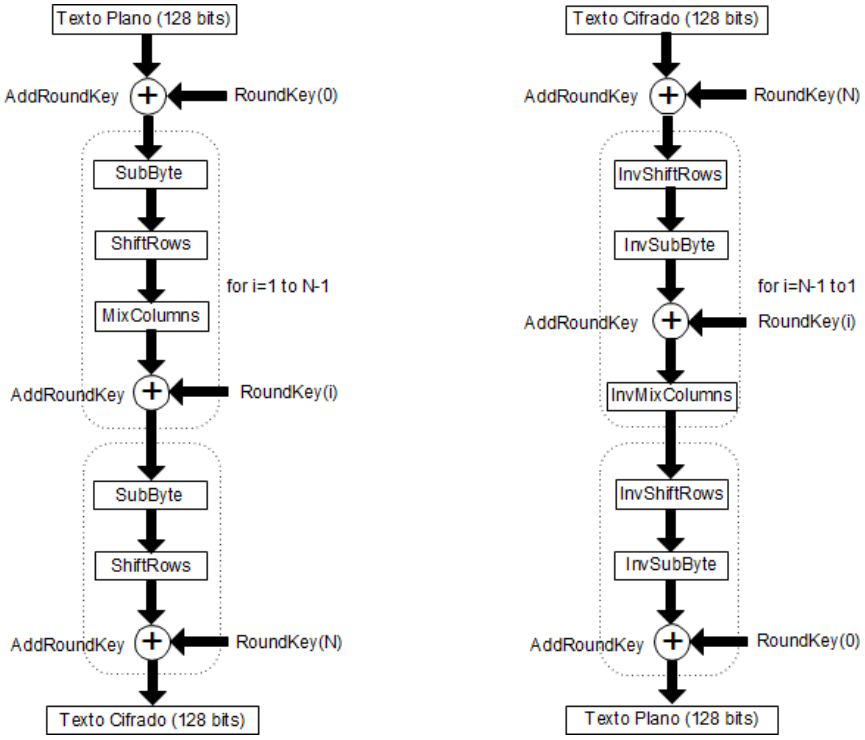

Figura 1 Diagrama a bloques del cifrado/descifrado AES.

La transformación `SubByte()` es realizada en forma independiente en cada byte de la matriz de estado utilizando una tabla de sustitución llamada caja-S (*S-Box*). La caja-S se construye calculando para cada byte el inverso multiplicativo sobre $GF(2^8)$ con el polinomio irreducible $m(x)$, y posteriormente se aplica una transformación *affine* sobre $GF(2)$, la cual consiste en la multiplicación por una matriz binaria y una suma vectorial. En el descifrado se utiliza una caja-S inversa que se obtiene al aplicar la transformación *affine* inversa seguida del cálculo del inverso multiplicativo para cada byte del arreglo de estado.

La transformación `ShiftRow()` se realiza sobre cada fila de la matriz de estado y consiste de desplazamientos circulares independientes en cada fila. La primera fila no se desplaza, mientras que la segunda, la tercera y la cuarta se desplazan a la izquierda un byte, dos bytes y tres bytes respectivamente. En la transformación inversa `InvShiftRows()`, la primera fila no cambia, en tanto que el resto de las filas se desplazan circularmente a la derecha uno, dos y tres bytes, de tal forma que se invierta la transformación directa.

La transformación `MixColumns()` se realiza independientemente sobre cada columna de la matriz de estado. Cada columna es considerada como un polinomio sobre $GF(2^8)$ y es multiplicada módulo x^4+1 con el polinomio fijo dado por:

$$a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\} \tag{2}$$

Utilizando notación matricial, se puede expresar la transformación `MixColumns()` como:

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

La transformación inversa `InvMixColumns()` se puede definir, en forma similar, como la multiplicación módulo $x^4 + 1$ con el polinomio fijo dado por:

$$a^{-1}(x) = \{0b\}x^3 + \{0d\}x^2 + \{09\}x + \{0e\} \tag{4}$$

La forma matricial correspondiente a la mezcla de columnas inversa `InvMixColumns()` está definida por:

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} \tag{5}$$

En la transformación `AddRoundKey()` se lleva a cabo una operación OR-Exclusiva (XOR) a nivel de bits entre el arreglo de estado y una llave de ronda de 128 bits. La transformación inversa es idéntica a la transformación directa, debido a que la operación XOR es su propia inversa.

En el descifrado (cifrado inverso), la secuencia de transformaciones difiere de la empleada durante el cifrado, tal como se muestra en la Figura 1. Sin embargo, algunas propiedades del algoritmo AES permiten un cifrado inverso equivalente que conserva la misma secuencia de transformaciones que el cifrado (reemplazando cada transformación por su respectiva transformación inversa). Para poder conservar la secuencia de transformaciones se requiere aplicar una transformación `MixColumns()` a cada llave de ronda antes de ser utilizada en la transformación `AddRoundKey()`. Como resultado de estos cambios se obtiene una estructura más eficiente al realizar el cifrado y descifrado. En la Figura 2 se muestra la secuencia de operaciones realizada en el cifrado inverso equivalente.

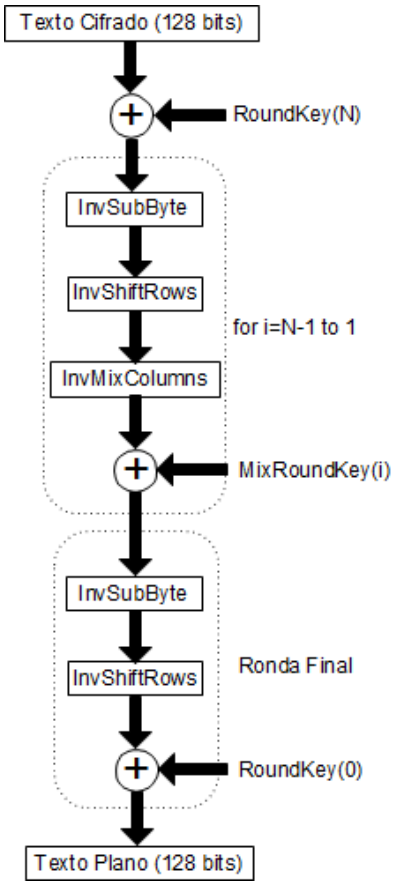

Figura 2 Diagrama a bloques del cifrado inverso equivalente.

III. Implementación de la arquitectura en hardware

El procesador NIOS II es una unidad central de procesamiento configurable que se define en un lenguaje descriptor de hardware y que puede ser implementada en los Arreglos de Compuertas Programables en Campo (FPGA, *Field Programmable Gate Array*) de la compañía Altera. El núcleo del procesador NIOS II es una arquitectura RISC (*Reduced Instruction Set Computer*) tipo Harvard que se caracteriza por un conjunto de instrucciones, ruta de datos y espacio de direccionamiento de 32 bits, y que cuenta con 32 registros de propósito general. Una característica distintiva de este procesador es la capacidad de permitirle al usuario definir instrucciones que se incorporan directamente a la Unidad Aritmética-Lógica (ALU, *Arithmetic Logic Unit*), como se muestra en la Figura 3, y optimizan el desempeño del sistema para alguna aplicación específica.

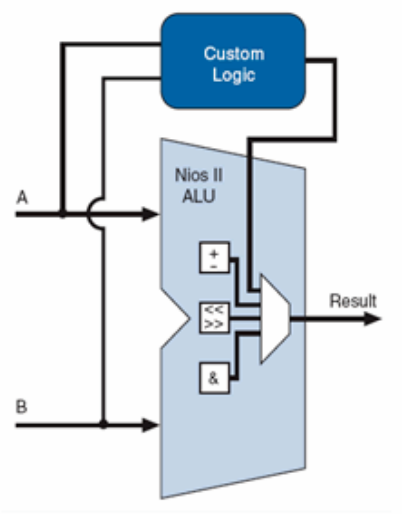

Figura 3 Instrucciones a medida en el NIOS II [?].

Las instrucciones a medida permiten adaptar el procesador NIOS II a las necesidades de una aplicación en particular [**?**]. En este trabajo se tomó como base la extensión de instrucciones para el cifrado AES propuesta por Tillich [**?**] para definir las instrucciones a medida (*Custom Logic*) en el procesador NIOS II. Para la implementación de las unidades funcionales utilizadas se realizó un análisis de diferentes técnicas para llevar a cabo las transformaciones `SubByte()` y `MixColumns()`.

En la Figura 4 se muestra el diagrama a bloques de la instrucción `sbox4s/isbox4s`. En esta se puede observar que la instrucción tiene 2 registros de 32 bits como operandos y el resultado se almacena en un registro de 32 bits. En esta instrucción, a diferencia de la transformación `SubByte()`, se realiza simultáneamente la sustitución de 4 bytes y adicionalmente se lleva a cabo una permutación de bytes entre los dos registros fuente.

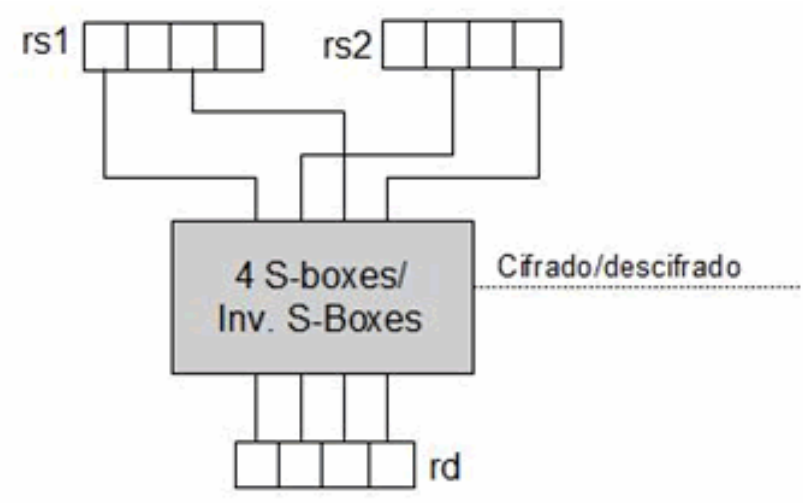

Figura 4 Instrucción `sbox4s/isbox4s`.

Las cajas S (*S-Boxes*) necesarias para la instrucción `sbox4s` se implementaron como una tabla de 256 bytes. De igual forma se procedió en el caso de la instrucción `isbox4s`. La principal ventaja de la instrucción `sbox4s/isbox4s` es que reduce el número de accesos a memoria en el algoritmo AES.

En la Figura 5 se muestra el diagrama a bloques de la instrucción `mixcol4s/imixcol4s`. Como se puede observar, la instrucción utiliza 2 registros de 32 bits como operandos y el resultado se almacena en un registro de 32 bits. En esta instrucción, al igual que en la transformación `MixColumns()`, se trabaja con una columna formada por 4 bytes; sin embargo, también se lleva a cabo una permutación de bytes que, en conjunto con la permutación realizada en la instrucción `sbox4s`, tiene como resultado final una transformación `ShiftRows()` implícita, con lo cual se logra reducir el tiempo de ejecución y el tamaño del código del algoritmo AES.

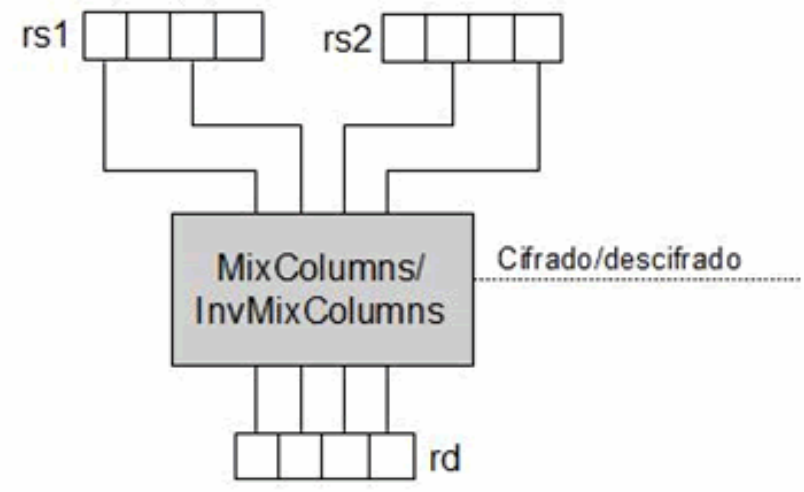

Figura 5 Instrucción `mixcol4s/imixcol4s`.

Para sintetizar la transformación `MixColumns()` se tomó en consideración que la ecuación (3) se puede expresar como [**?**]:

$$
\begin{aligned}
s'_{0,c} &= \{02\} \cdot s_{0,c} \oplus \{03\} \cdot s_{1,c} \oplus s_{2,c} \oplus s_{3,c} \\
s'_{1,c} &= s_{0,c} \oplus \{02\} \cdot s_{1,c} \oplus \{03\} \cdot s_{2,c} \oplus s_{3,c} \\
s'_{2,c} &= s_{0,c} \oplus s_{1,c} \oplus \{02\} \cdot s_{2,c} \oplus \{03\} \cdot s_{3,c} \\
s'_{3,c} &= \{03\} \cdot s_{0,c} \oplus s_{1,c} \oplus s_{2,c} \oplus \{02\} \cdot s_{3,c}
\end{aligned} \tag{6}
$$

De acuerdo a (6), la implementación de la instrucción `mixcol4s` solo requiere multiplicar por la constante $\{02\}$. Como cada byte en la transformación `MixColumns()` se puede expresar en forma polinomial, el producto por la constante $\{02\}$ está dado por la expresión:

$$
\{02\} \cdot b = \begin{cases} b \ll 1 & \text{si } b_7 = 0 \\ (b \ll 1) \oplus \{1b\} & \text{si } b_7 = 1 \end{cases} \tag{7}
$$

La transformación inversa `InvMixColumns()` se puede expresar como:

$$
\begin{aligned}
s_{0,c} &= \{0e\} \cdot s'_{0,c} \oplus \{0b\} \cdot s'_{1,c} \oplus \{0d\} \cdot s'_{2,c} \oplus \{09\} \cdot s'_{3,c} \\
s_{1,c} &= \{09\} \cdot s'_{0,c} \oplus \{0e\} \cdot s'_{1,c} \oplus \{0b\} \cdot s'_{2,c} \oplus \{0d\} \cdot s'_{3,c} \\
s_{2,c} &= \{0d\} \cdot s'_{0,c} \oplus \{09\} \cdot s'_{1,c} \oplus \{0e\} \cdot s'_{2,c} \oplus \{0b\} \cdot s'_{3,c} \\
s_{3,c} &= \{0b\} \cdot s'_{0,c} \oplus \{0d\} \cdot s'_{1,c} \oplus \{09\} \cdot s'_{2,c} \oplus \{0e\} \cdot s'_{3,c}
\end{aligned} \tag{8}
$$

La ecuación (8) implica que las operaciones realizadas en la transformación `MixColumns()` pueden ser reutilizadas en la implementación de la transformación inversa. La transformación `InvMixColumns()` requiere de la multiplicación por las constantes $\{02\}$ y $\{04\}$. La multiplicación por $\{04\}$ se implementó mediante la ecuación polinomial:

$$
\{04\} \cdot b = \{02\} \cdot (\{02\} \cdot b) \tag{9}
$$

IV. Validación y Resultados

La implementación del sistema se realizó utilizando un sistema de desarrollo DE2 basado en un FPGA Cyclone II EP2C35672C6 de Altera. Como herramienta de síntesis se utilizó el ambiente de desarrollo Quartus II Web Edition. La descripción en hardware de las unidades funcionales para las instrucciones `sbox4s` y `mixcol4s` se realizó utilizando Verilog HDL.

En la Figura 6 se muestra el resultado obtenido de la simulación de la transformación `MixColumns()`. En esta, el valor en `dataa` corresponde al vector columna en (3) y representa la entrada al bloque que realiza la transformación; en tanto, `result` corresponde a la salida del bloque y representa el resultado de la mezcla de columnas. Los valores obtenidos se validaron con los datos mostrados en el apéndice B del FIPS-197.

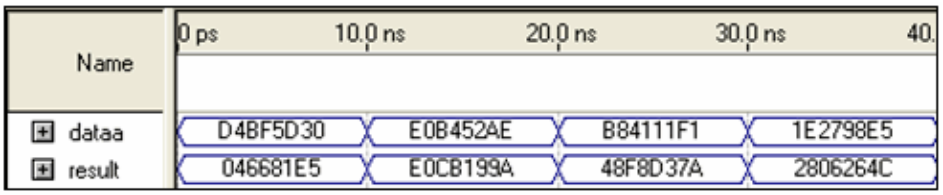

Figura 6 Resultados de la transformación `MixColumns()`.

En la Figura 7 se muestran los resultados de la transformación `SubByte()` al aplicarse a una columna formada por 4 bytes. En esta figura el valor en `dataa` representa el dato en un registro de 32 bits que almacena 4 bytes, mientras que el valor de `result` es el resultado de la sustitución correspondiente a cada byte. Los valores obtenidos corresponden a los valores esperados de acuerdo al FIPS-197.

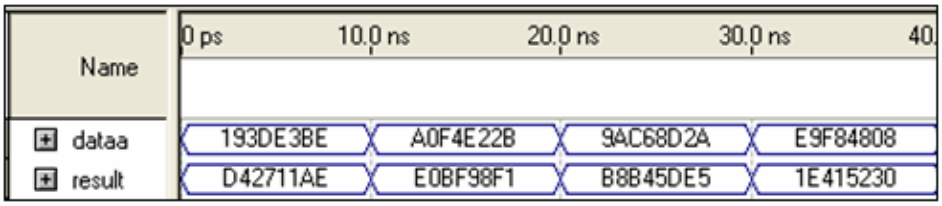

Figura 7 Resultados de la transformación `SubByte()`.

Para sintetizar el procesador NIOS II se utilizó el programa SOPC Builder, el cual permite configurar el núcleo del procesador y agregar diferentes periféricos y tipos de memoria para formar un sistema completo en un circuito programable (SOPC, *System On a Programmable Chip*). Esta herramienta también se puede utilizar para agregar instrucciones a medida (*Custom Instructions*) en el procesador NIOS II.

Después de sintetizar y agregar a la ALU las unidades funcionales para las instrucciones `sbox4s` y `mixcol4s`, se programó el algoritmo AES utilizando el lenguaje de programación C, para ello se utilizó el entorno de desarrollo NIOS II IDE. En la Figura 8 se muestra el programa en C que realiza una ronda completa del algoritmo AES utilizando las instrucciones definidas. En este fragmento de código se pueden observar las llamadas a las funciones `SBOX4S()` y `MIXCOL4S()`, las cuales en realidad son macros definidas en un archivo de cabecera y son las encargadas de invocar la función para dar soporte a las instrucciones `sbox4s` y `mixcol4s`.

```
for (k=1;k<=9;k++){
// SubBytes con ShiftRow implicito
  t0=SBOX4S(0,s0,s1);
  t1=SBOX4S(0,s1,s2);
  t2=SBOX4S(0,s2,s3);
  t3=SBOX4S(0,s3,s0);
// MixColumns con ShiftRow implicito
  s0=MIXCOL4S(0,t0,t2);
  s1=MIXCOL4S(0,t1,t3);
  s2=MIXCOL4S(0,t2,t0);
  s3=MIXCOL4S(0,t3,t1);
// AddRoundKey
  s0=s0^roundkey[k][0];
  s1=s1^roundkey[k][1];
  s2=s2^roundkey[k][2];
  s3=s3^roundkey[k][3];
}
```

Figura 8 Código en lenguaje C para una ronda AES.

Cada una de las funciones `SBOX4S()` y `MIXCOL4S()` tiene 3 argumentos; cuando el primer argumento toma el valor «0», la función realizará la transformación directa; si el valor es «1», se llevará a cabo la transformación

inversa. Los argumentos restantes corresponden a los operandos de 32 bits indicados en las Figuras 4 y 5.

En la Figura ?? se muestra la consola que utiliza el ambiente NIOS II IDE para comunicarse con la tarjeta de desarrollo DE2 a través de una interfaz USB. En esta figura se puede observar el resultado del cifrado del bloque de datos `0x3243f6a8885a308d313198a2e0370734` cuando se utiliza una clave de cifrado de 128 bits definida por `0x2b7e151628aed2a6abf7158809cf4f3c`.

De igual forma se puede programar el descifrado utilizando las instrucciones definidas mediante la estructura de cifrado inverso equivalente. Por otra parte, las instrucciones a medida también se pueden emplear en el lenguaje ensamblador para el procesador NIOS II, a través de la instrucción `custom`, la cual permite acceder hasta 256 instrucciones definidas por el usuario.

V. Conclusiones

En este trabajo se llevó a cabo el diseño hardware/software para la implementación del algoritmo de cifrado AES. La sustitución de bytes y la mezcla de columnas del algoritmo se desarrollaron en hardware, ya que estas transformaciones representan las operaciones que mayor tiempo consumen en la implementación en software del algoritmo AES. Al integrar las unidades funcionales realizadas en un procesador NIOS II como lógica adicional a la ALU se extiende el conjunto básico de instrucciones, con lo cual se logra reducir el tiempo de ejecución para cada transformación a tan solo un ciclo de reloj. Este enfoque permite combinar la flexibilidad que ofrece el software con el desempeño del hardware dedicado. Es importante destacar que la metodología seguida se puede extender a otros algoritmos de cifrado.

Referencias

[1] National Institute of Standards and Technology (NIST). *FIPS-197: Advanced Encryption Standard*. 2001.

[2] P. Chodowiec y K. Gaj. «Very compact FPGA implementation of the AES algorithm». *Cryptographic Hardware and Embedded Systems --- CHES 2003*, Cologne, Alemania, sep. 2003.

[3] T. Good. «Very small FPGA application-specific instruction processor for AES». *IEEE Trans. on Circuits and Systems*, vol. 53, no. 7, jul. 2006.

[4] P. Hämäläinen, J. Heikkinen, M. Hännikäinen y T. Hämäläinen. «Design of transport triggered architecture processors for wireless encryption». *8th Euromicro Conference on Digital System Design*, ago. 2005.

[5] H. W. Kim y S. Lee. «Design and implementation of a private and public key crypto processor and its application to a security system». *IEEE Transactions on Consumer Electronics*, vol. 50, no. 1, feb. 2004.

[6] F. Dacâncio Pereira, E. D. Moreno Ordonez y R. Barros Chiaramonte. «VLIW cryptoprocessor: Architecture and performance in FPGAs». *International Journal of Computer Science and Network Security*, vol. 6, no. 8A, ago. 2006.

[7] D. Oliva, R. Buchty y N. Heintze. «AES and the Cryptonite crypto processor». *International Conference on Compiler Architecture and Synthesis for Embedded Systems --- CASES 2003*, oct. 2003.

[8] L. Wu, C. Weaver y T. Austin. «CryptoManiac: A fast flexible architecture for secure communication». *28th Annual International Symposium on Computer Architecture --- ISCA-2001*, jun. 2001.

[9] O. Harrison y J. Waldron. «AES encryption implementation and analysis on commodity graphics processing units». *Cryptographic Hardware and Embedded Systems --- CHES 2007*, sep. 2007.

[10] S. A. Manavski. «CUDA compatible GPU as an efficient hardware accelerator for AES cryptography». *IEEE International Conference on Signal Processing and Communication --- ICSPC 2007*, nov. 2007.

[11] A. J. Elbirt. «Fast and efficient implementation of AES via instruction set extensions». *21st International Conference on Advanced Information Networking and Applications Workshops*, may. 2007.

[12] S. Tillich y J. Großschädl. «Instruction set extensions for efficient AES implementation on 32-bit processors». *Cryptographic Hardware and Embedded Systems --- CHES 2006*, oct. 2006.

[13] Altera. *NIOS II Custom Instructions User Guide*. Mayo 2008.