FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Implementação em Verilog do algoritmo de cifra AES-CTR para aplicações HDMI 2.0

Hugo Miguel Teixeira Fernandes

PREPARAÇÃO DA DISSERTAÇÃO

PREPARAÇÃO DA DISSERTAÇÃO

Orientador na Feup: Prof. Dr. João Canas Ferreira

Orientador na Synopsys: Eng. Luís Laranjeira

17 de Fevereiro de 2014

Implementação em Verilog do algoritmo de cifra AES-CTR para aplicações HDMI 2.0

Hugo Miguel Teixeira Fernandes

PREPARAÇÃO DA DISSERTAÇÃO

17 de Fevereiro de 2014

Conteúdo

1 Introdução **1**
- 1.1 Enquadramento . . . 1
- 1.2 Sobre a Synopsys . . . 2
- 1.3 Objetivos . . . 2

2 Estado da Arte **5**
- 2.1 High Digital Content Protection (HDCP) . . . 5
- 2.2 Advanced Encryption Standard (AES) . . . 6
- 2.3 Operações Aritméticas Básicas . . . 7
- 2.4 Estrutura Interna do AES . . . 8
  - 2.4.1 SubBytes . . . 9
  - 2.4.2 ShiftRows . . . 10
  - 2.4.3 MixColumns . . . 11
  - 2.4.4 AddRoundKey . . . 12
  - 2.4.5 Key Expansion . . . 12
- 2.5 AES - Counter Mode . . . 13

3 Proposta de Trabalho **15**
- 3.1 Implementação . . . 15
- 3.2 Modelo de Alto Nível . . . 17
  - 3.2.1 SubBytes . . . 18
  - 3.2.2 Key Expansion . . . 18
  - 3.2.3 MixColumns . . . 20
  - 3.2.4 ShifRows . . . 20
  - 3.2.5 AddRoundKey . . . 20
- 3.3 Plano de trabalho . . . 21

Referências **23**

Lista de Figuras

2.1 Estrutura da cifra HDCP 6
2.2 Estrutura geral do algoritmo AES[2] 8
2.3 Transformação Affine[2] 9
2.4 Transformação SubBytes[2] 10
2.5 Tabela de substituição Sbox[2] 10
2.6 Transformação ShiftRows[3] 11
2.7 Transformação MixColumns[2] 11
2.8 Algoritmo de expansão da chave[1] 13
2.9 Counter Mode[4] 14

3.1 Implementações S-box realizadas em Alireza Hodjat (2006) 16
3.2 Area vs Taxa de transferencia com Key Expansion *online* 16
3.3 Area vs Taxa de transferencia com Key Expansion *offline* 17
3.4 Resultados obtidos por Hodjat e Verbauwhede (2004) 18
3.5 Diagrama de Gantt 21

Abreviaturas e Símbolos

BD	Blu-ray
DVD	Digital versatile disc
HDCP	High-Bandwith Digital Content Protection
HDMI	High-Definition Multimedia Interface
DVI	Digital Visual Interface
AES	Advanced encryption standard
NIST	National Institute of standards and technology
FPGA	Field Programmable Gate Array
HD	High Definition
TMDS	Transition-minimizaed differential signaling
CEC	COnsumer Electronics Control
CTR	Counter mode
GF	Galois Fields
k_s	Chave de sessão
UHD	Ultra High Definition

Capítulo 1

Introdução

Esta dissertação foi proposta pela SYNOPSYS Portugal e é realizada no âmbito do Mestrado Integrado em Engenharia Eletrotécnica e de Computadores da Faculdade de Engenharia da Universidade do Porto.

1.1 Enquadramento

Os conteúdos audiovisuais estão cada vez mais a ser distribuídos aos consumidores em formato digital, através da Internet, redes de satélites, reprodutores *DVD(digital versatile disc)* ou *BD (blu-ray)*. Esta vasta disponibilidade de conteúdo digital fez com que os fornecedores do mesmo se preocupassem cada vez mais com a copia não autorizada e o seu uso. Como resultado os fabricantes de conteúdo, fornecedores e os fabricantes de dispositivos eletrónicos implementaram uma variedade de técnicas de proteção de conteúdos, que protegem o acesso ao conteúdo multimédia distribuído através de diferentes meios de comunicação. O *HDCP (High-bandwidth Digital Content Protection)* é uma parte desta cadeia de proteção. Este protege o ultimo estágio do processo de distribuição, a transmissão do conteúdo entre um dipositivo transmissor como por exemplo, um leitor de *DVD* e recetor como por exemplo, um Televisor. Tendo sido desenvolvido pela Intel, o sistema foi feito para parar de transmitir conteúdo em dispositivos não autorizados ou que foram modificados para copiar o conteúdo *HDCP*. Antes de enviar, o dispositivo transmissor verifica se o recetor esta autorizado a receber o conteúdo, se for autorizado, o transmissor encripta o conteúdo para prevenir a copia do conteúdo enquanto este é transmitido. Os interfaces compliantes com o *HDCP* são o *HDMI(High-Definition Multimedia Interface), DVI e DisplayPort*. Com uma base instalada superior a dois mil milhões de dispositivos de consumo, sendo de longe a interface com maior adoção para aplicações *"home entertainment"* e *"mobile multimedia"*, o interface *HDMI* tornou-se o standard substituindo o *DVI*. O desenvolvimento do *HDMI* 1.0 foi iniciado em 2002 com o objetivo de criar um conector AV que fosse retrocompativel com o *DVI*. Na altura o standard *DVI* estava a ser usado em televisores *HD*. O *HDMI* 1.0 foi desenhado para melhorar o *DVI* ao usar um conector mais pequeno com suporte para áudio e suporte melhorado para *"YCbCr"* e funções de controlo dos dispositivos *(CEC)*IDC (2006).

Desde então a interface *HDMI* tem evoluido, estando neste momento na especificação 2.0. A cada nova especificação foram sendo acrescentadas funcionalidades e aumentando a largura de banda. Quando foi lançada a especificação *HDMI* 1.0 esta suportava uma taxa de transferência **TMDS** máximo de 4.95 Gbits/s tendo evoluído para 18Gbit/s na especificação HDMI 2.0. A especificação *HDMI* 2.0 trouxe a possibilidade de transporte de resoluções mais elevadas, tais como *Ultra HD (Ultra High Definition)* (4k x 2k a 60Hz). O aumento da taxa de transferência trouxe a necessidade da implementação em hardware de um algoritmo de cifra AES-CTR capaz de suportar uma taxa de transferencia de 18Gbit/s. Esta dissertação realizada em conjunto com a SYNOPSYS tem como objectivo a implementação de um algoritmo de cifra AES-CTR capaz de obter a taxa de transferencia acima referida, assim como uma optimização a nivel de area e numero de gates de modo a se conseguir reduzir o tamanho do die e consequentes custos de produção.PROTECTION (2008).

1.2 Sobre a Synopsys

SYNOPSYS é a empresa líder de mercado em propriedade intelectual de semicondutores (IP) e automação de desenho eletrónico(EDA). Por mais de 25 anos , a SYNOPSYS tem sido o motor na aceleração da inovação eletrónica com engenheiros em todo o mundo a usarem a sua tecnologia para projetar e criar com sucesso milhares de milhões de chips e sistemas eletrónicos de que as pessoas dependem no dia a dia. Foi fundada em 1986 pelo Dr. Aart de Geus e uma equipa de engenheiros do Centro de Micro-eletrónica da General Electrics na Carolina do Norte. A empresa foi pioneira na aplicação comercial de síntese lógica que já foi adotada por todas as grandes empresas de design de semicondutores do mundo. Esta tecnologia proporcionou um salto exponencial de produtividade no projeto de desenho de circuitos integrados (IC), permitindo aos engenheiros especificar a funcionalidade dos chips a um nível mais alto de abstração. Sem essa tecnologia, os projetos complexos de hoje não seriam possíveis. A empresa abastece o mercado de eletrónica global com software, propriedade intelectual(IP), serviços utilizados no design de circuitos integrados, verificação e fabrico. Tudo isto é conseguido fornecendo um portfólio integrado de implementação, verificação, fabrico, e *field-programmable gate array* (FPGA) para abordar os desafios na produção de circuitos integrados (IC), como o consumo, verificação de *software-to-silicon* e tempo de resultadosSynopsys (2014).

1.3 Objetivos

A inclusão da cifra AES-CTR em aplicações HDMI 2.0 acarreta um objetivo de desempenho muito elevado, uma vez que a cifra é aplicada no caminho de dados áudio/vídeo de 24 bits, com uma cadência de até 600MHz.Pretende-se por isso desenvolver uma implementação que permita síntese para ASIC em tecnologia 28nm, cumprindo os objetivos de desempenho mencionados. Esta dissertação tem como objetivos o desenvolvimento, implementação e verificação do algoritmo de cifra "Advanced Encryption Standard"(AES) em modo de operação Counter(CTR). A

implementação será realizada em linguagem de descrição de hardware Verilog e terá de ser sintetizável em tecnologia 28nm. O objetivo de desempenho é 600Mhz em FPGA Xilinx Virtex 6, com processamento de 24 bits por ciclo de relógio.

Capítulo 2

Estado da Arte

Neste capitulo será feita uma introdução ao *HDCP* e mais concretamente a encriptação do conteúdo, o algoritmo usado e o seu funcionamento interno.

2.1 High Digital Content Protection (HDCP)

Com a introdução e adoção de *HDTV(High-definition Television)* e a sua capacidade de suportar resoluções muito elevadas, alguns segmentos da industria aumentaram as suas preocupações em relação aos riscos de pirataria, principalmente devido a cópia não autorizada de filmes em alta resolução e a consequente perda de rendimentos que dai advém, fez com que se tomassem medidas para a sua prevenção. Apesar de já existirem métodos de proteção contra cópia não autorizada do meio físico, por exemplo de *DVD* e *BD*. Tornou-se necessário proteger também a transmissão do seu conteúdo durante a reprodução de modo a não poder ser copiado durante a reprodução. O *HDCP* foi desenvolvido para proteger a transmissão de conteúdo audiovisual entre um transmissor e recetor. O *HDCP* recorre ao algoritmo de cifra *AES-CTR* para a cifragem e decifragem. O processo decorre da seguinte forma, após o processo de autenticação entre o transmissor e o recetor tenha sido concluído com sucesso, é iniciada uma sessão de *Key exchange* onde o transmissor gera uma chave de sessão de 128 bits e um vetor de entrada pseudo aleatório *IV* de 64 bits, encripta esses valores e os envia para o recetor onde são recuperados pelo mesmo. Após esta troca, o transmissor espera pelo menos 200ms até ativar a encriptação e dar inicio a transmissão de conteúdo encriptado. A encriptação é feita usando o algoritmo de cifra AES no modo *counter* que será explicado ao pormenor nas secções seguintes. Na figura 2.1 é apresentada a estrutura da cifra *HDCP*. Em que a chave de cifragem corresponde a um *XOR* entre k_s, que corresponde a chave de sessão e lc_{128} que é uma constante global de 128 bits partilhada por todos os dispositivos *HDCP*. O bloco de entrada de 128 bits a ser cifrado corresponde a uma concatenação entre o vetor de entrada de 64 bits *IV* e o bloco *inputCtr* também de 64 bits. Por sua vez, $inpuCtr = FrameNumber||DataNumber$, em que $FrameNumber$ é um valor de 38 bits que indica o numero de tramas encriptadas desde o inicio da encriptação *HDCP*. O valor de $FrameNumber$

é incrementado por 1 a cada frame transmitida. *DataNumber* é um valor de 26 bits que é incrementado a cada geração de um bloco *keystream* de 128 bits. O valor do *inputCtr* é inicializado a zero quando a encriptação é ativada pela primeira vez após o inicio de sessão. *keystream* corresponde ao bloco de saída da cifra, em que é realizado um *XOR* com *InputData* que correspondem aos dados a serem encriptados e temos como saída *EncryptedOutput* que corresponde aos dados encriptadosPROTECTION (2008)Protection (2013).

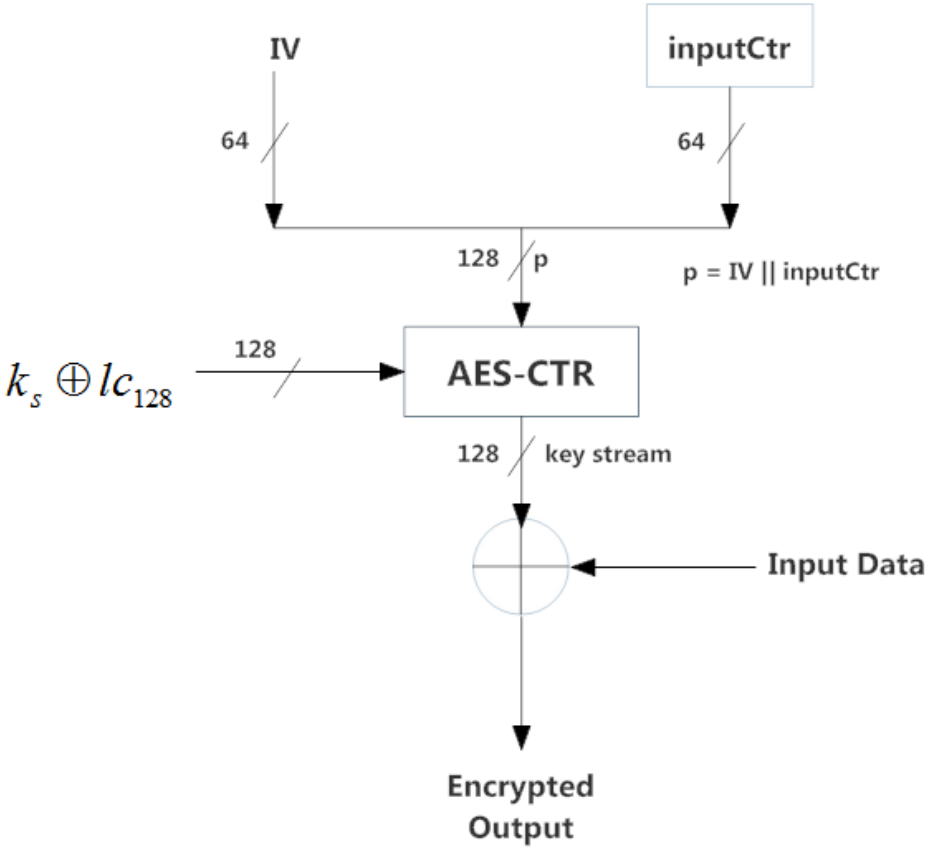

Figura 2.1: Estrutura da cifra HDCP

2.2 Advanced Encryption Standard (AES)

Em 1997 a National Institute of standards and technology(NIST) abriu um concurso para um novo standard de encriptação Avançada(AES).

Os algoritmos candidatos tinham de preencher os seguintes requisitos:

- Cifra de bloco com tamanho de 128 bits.
- Suportar três tamanhos de chave diferentes (128, 192 e 256) bits.
- Segurança relativamente aos outros algoritmos submetidos.

- Eficiência em hardware e software.

Em 2000 o *NIST* anunciou a escolha do bloco *Rijndael* como a cifra AES, tornando-se o standard em 26 de Maio de 2002. Esta cifra é obrigatória em muitos standards da industria tais como *HDCP*, *TLS*, standard de encriptação *wifi IEEE 802.11i*, entre outros. A encriptação AES é uma cifra de bloco simétrica em que o comprimento do bloco de dados é de 128 bits e o comprimento da chave de cifragem pode ser 128, 192 ou 256 bits. O AES "opera"numa matriz de bytes 4x4 denominada de matriz de estado e a maioria dos cálculos do AES são feitos em operações aritméticas básicas GF(2^8)(campos finitos)

A encriptação é feita em *rounds*. O numero de *rounds* é definido pelo comprimento da chave e pode ser 10, 12 ou 14 conforme a chave seja de 128, 192 e 256 bits. Cada *round* com exceção da ultima, é composta por quatro transformações: SubBytes, ShifRows, MixColumns e AddRoundKey as quais serão explicadas na secção seguinte. Na matriz de estado os bytes são orientados por coluna, isto é, os primeiros quatro bytes do bloco de entrada são dispostos na primeira coluna, os quatro bytes seguintes na segunda coluna e assim consecutivamente. C. Paar and J. Pelzl

A estrutura geral do algoritmo AES , é apresentada na figura 2.2. Na figura pode-se ver que o *"plaintext"*, dados a serem cifrados com o tamanho de 16 bytes, são colocados na matriz de estado 4x4, a qual é feita uma transformação inicial chamada de *"key whitening"* que consiste numa transformação AddRoundKey com a chave original, após a qual se dá o inicio das rounds que são todas exatamente iguais com a exceção da ultima em que a transformação MixColumns não é feita, dando depois origem ao *"Ciphertext"* que corresponde aos dados cifrados.

2.3 Operações Aritméticas Básicas

No algoritmo AES, todos os bytes são interpretados como sendo elementos de um campo finito que podem ser representados por uma descrição polinomial, como a expressa pela equação

$$b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x^1 + b_0$$

A adição de dois elementos em campos finitos é conseguida, adicionando os coeficientes das respetivas potências através de uma operação XOR modulo 2 o que corresponde a um XOR bitwise. A adição e a subtração de polinómios produzem o mesmo resultado.

No caso da multiplicação, esta é dada pelo resto da divisão do produto de dois polinómios por um polinómio irredutível, ou , seja, um polinómio de grau 8 que é divisível por 1 e por ele próprio.

No algoritmo AES, o polinómio irredutível é o da expressão seguinte:

$$m(x) = x^8 + x^4 + x^3 + x + 1$$

A redução modular por $m(x)$ assegura que o resultado será sempre um polinómio binário de grau inferior a oito, e que pode ser representado por um byte.PUBS (2001)

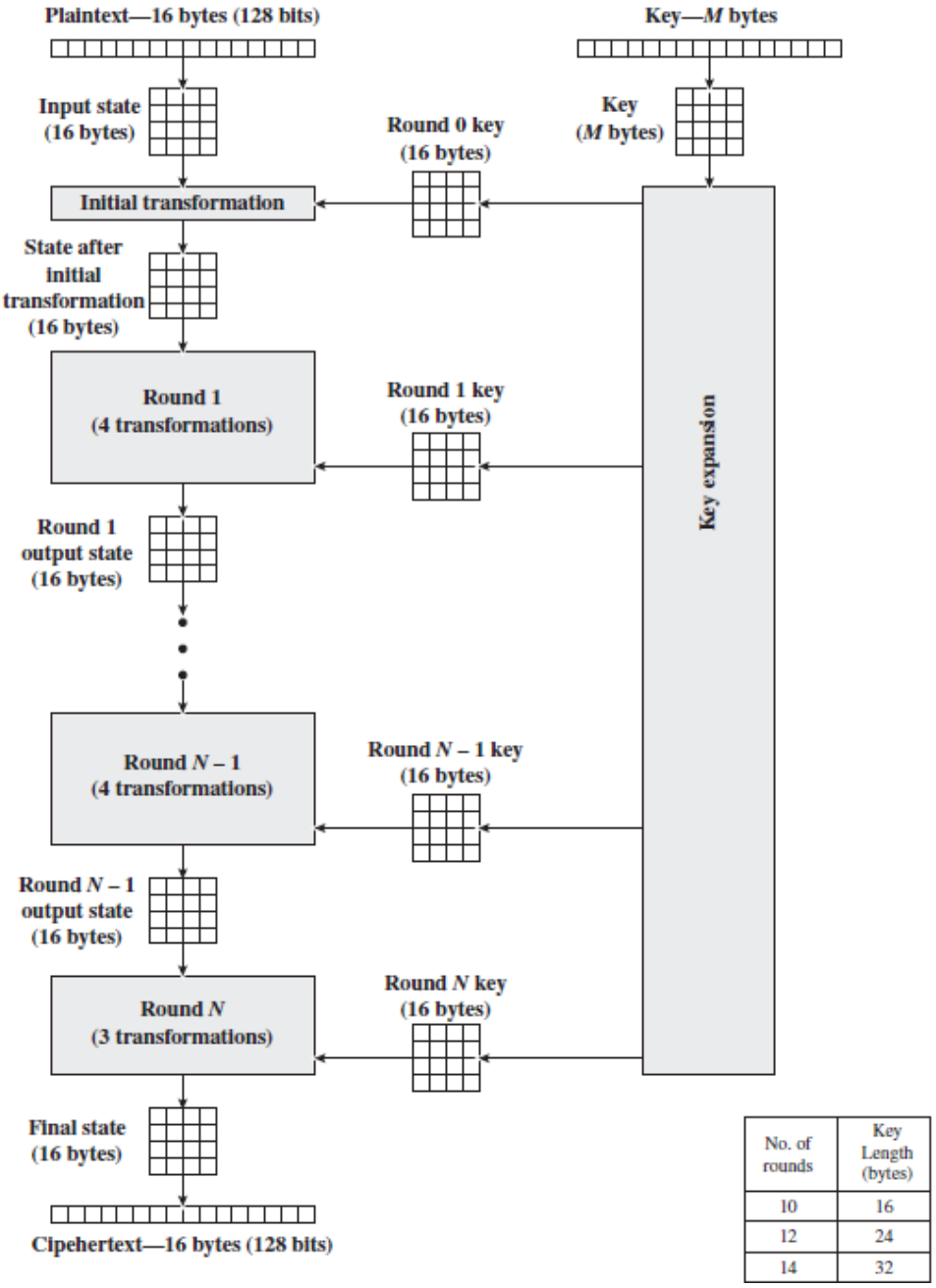

Figura 2.2: Estrutura geral do algoritmo AES[2]

2.4 Estrutura Interna do AES

Nesta secção é analisada a estrutura interna do AES.

2.4.1 SubBytes

A transformação SubBytes, consiste numa transformação não linear em que cada byte da matriz de estado é substituído por outro através de uma tabela de referência à qual se dá o nome de *S-box*. A tabela de substituição ou S-box representada na figura 2.4 é invertível e é construida pela composição de duas operações, a multiplicativa inversa em GF(2^8) e a transformação *Affine* em GF(2), esta ultima é dada por:

$$b'_i = b_i \oplus b_{(i+4)mod8} \oplus b_{(i+5)mod8} \oplus b_{(i+6)mod8} \oplus b_{(i+7)mod8} \oplus c_i$$

para $0 \leq i < 8$, onde b_i é o i^{th} bit do byte e c_i é o i^{th} bit do byte c com o valor hexadecimal63. Esta transformação pode ser representada no formato de matriz, como mostra a figura 2.3.

$$\begin{bmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}.$$

Figura 2.3: Transformação Affine[2]

A tabela S-box pode ser vista como uma tabela 16 x 16 com 1 byte de entrada e 1 byte de saída. A substituição é feita da seguinte forma, os quatro bits mais significativos do byte a ser substituído representam a posição da linha na tabela **Sbox** e os quatro menos significativos a posição na coluna. Estes valores servem de índices da tabela para selecionar um byte da tabela. A tabela esta representada na figura 2.5. Como exemplo, se tivermos como byte de entrada o valor hexadecimal 25 em que se refere a linha 2, coluna 5 na Sbox, a sua saída será o valor hexadecimal de C2.

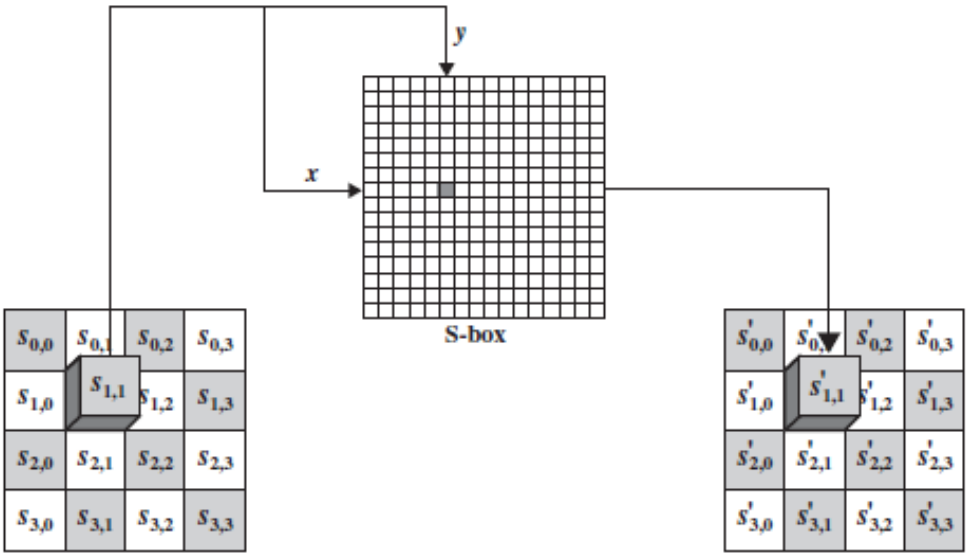

Figura 2.4: Transformação SubBytes[2]

		y														
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
x	0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
	1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
	2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
	3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
	4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
	5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
	6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
	7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
	8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
	9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
	A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
	B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
	C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
	D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
	E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
	F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

Figura 2.5: Tabela de substituição Sbox[2]

2.4.2 ShiftRows

A transformação ShiftRows, consiste numa permutação simples, onde a primeira linha se mantém inalterada, a segunda linha da matriz é rodada a esquerda por um byte, a terceira linha é rodada a esquerda de dois bytes e a quarta linha de três bytes. O objetivo desta é de aumentar a propriedade de difusão. A transformação ShiftRows é ilustrada na figura 2.6.

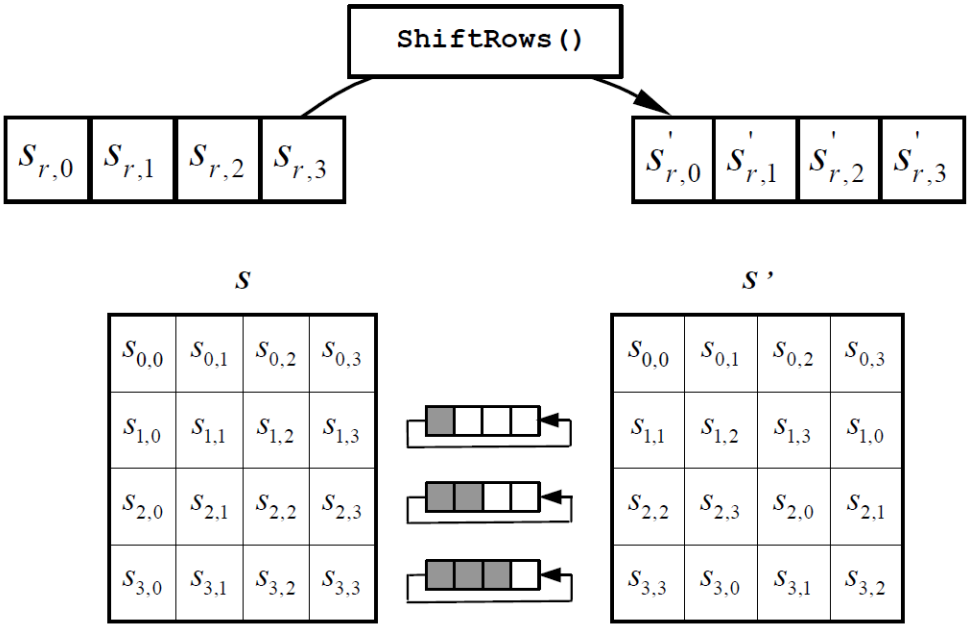

Figura 2.6: Transformação ShiftRows[3]

2.4.3 MixColumns

A transformação MixColumns consiste numa substituição que faz uso da aritmética sobre GF(2^8) e opera em cada coluna da matriz de estado individualmente. Cada coluna de 4 bytes é considerada como um vetor e é multiplicada por uma matriz 4x4. A matriz contem valores constantes. A figura 2.7 apresenta a transformação **MixColumns**.

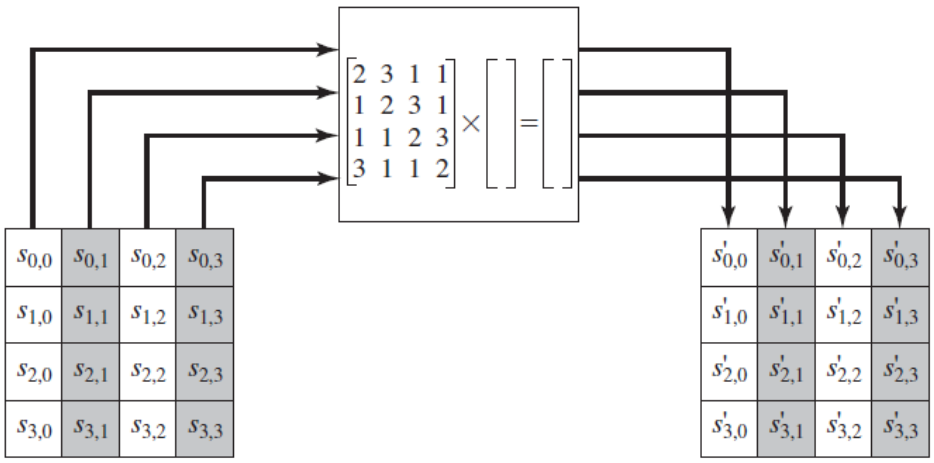

Figura 2.7: Transformação MixColumns[2]

Cada elemento da matriz de saída é a soma dos produtos dos elementos de uma linha e uma coluna. A transformação de uma única coluna da matriz de estado pode ser expressa pelas equações (2.1):

$$S'_{0,j} = (2 \cdot S_{0,j}) \oplus (3 \cdot S_{1,j}) \oplus S_{2,j} \oplus S_{3,j}$$

$$S'_{1,j} = S_{0,j} \oplus (2 \cdot S_{1,j}) \oplus (3 \cdot S_{2,j}) \oplus S_{3,j}$$

$$S'_{2,j} = S_{0,j} \oplus S_{1,j} \oplus (2 \cdot S_{2,j}) \oplus (3 \cdot S_{3,j})$$

$$S'_{3,j} = (3 \cdot S_{0,j}) \oplus S_{1,j} \oplus S_{2,j} \oplus (2 \cdot S_{3,j})$$

Como se pode observar pelas equações (2.1), cada byte de entrada influencia quatro bytes de saída, esta operação é o maior elemento difusor da cifra *AES*. A combinação de ShiftRows e MixColumns torna possível de que ao fim de apenas três rondas cada byte da matriz dependa de todos os 16 bytes do *plaintext*.

2.4.4 AddRoundKey

Esta transformação consiste apenas na aplicação de uma operação lógica XOR bit a bit do bloco da matriz de estado com os 128 bits da chave expandida correspondente a essa *round*.

2.4.5 Key Expansion

O algoritmo de expansão de chave tem como entrada a chave e são calculadas 10 sub-chaves, sendo depois guardadas a chave e as subchaves num array de 44 words. A chave e subchaves são posteriormente usadas em cada round na transformação AddRoundKey. As sub-chaves são calculadas como mostra a figura 2.8, em que a word mais a esquerda de cada roundkey $W[4i]$, é calculada da seguinte forma , sendo $i = 1, ..., 10$:

$$W[4i] = W[4(i-1) + g(W[4i-1])]$$

onde g() é uma função não linear com quatro bytes de entrada e saída. As restantes três words de cada sub-chave são calculadas como:

$$W[4i+j] = W[4i+j-1] + W[4(i-1)+j]$$

onde$i = 1, ..., 10$e $j = 1, 2, 3$. A função g() como mostra a figura 2.8, faz uma rotação de um byte para a esquerda da word de entrada, executa uma substituição *bytewise SubBytes* e adiciona uma constante *round coefficient(RC)* ao byte mais a esquerda na função g(). O *round coefficient* varia de ronda para ronda de acordo com:

$$RC[1] = x^0 = (00000001)_2$$

$$RC[2] = x^1 = (00000010)_2$$

$$RC[3] = x^2 = (00000100)_2$$

$$\vdots$$

$$RC[10] = x^9 = (00110110)_2$$

A finalidade da função g() é a de adicionar não linearidade e remover simetria as chaves.

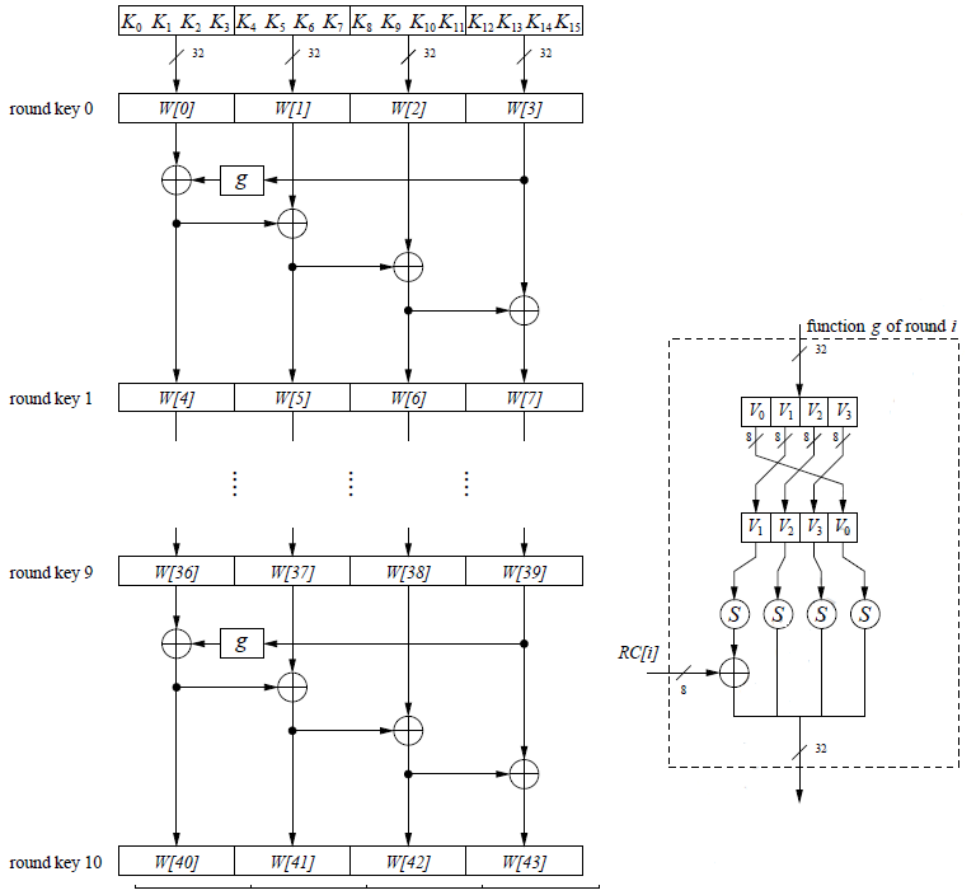

Figura 2.8: Algoritmo de expansão da chave[1]

2.5 AES - Counter Mode

O *counter mode* é um modo de funcionamento da cifra AES em que podemos computar o bloco AES a priori sem a presença dos dados a serem cifrados. Estes blocos são chamados de *counters*, sendo no final adicionados os dados a serem cifrados através de um *bitwise* XOR entre o *counter* e os dados, resultando dai os dados cifrados como demonstra a figura 2.9

A sequência de *counters* devem ter a propriedade de que cada bloco da sequência(*counter*)seja diferente, e esta condição tem de ser verificada para todas as mensagens que são encriptadas com a mesma chave. Ou seja, todos os *counters* têm de ter valor diferente.

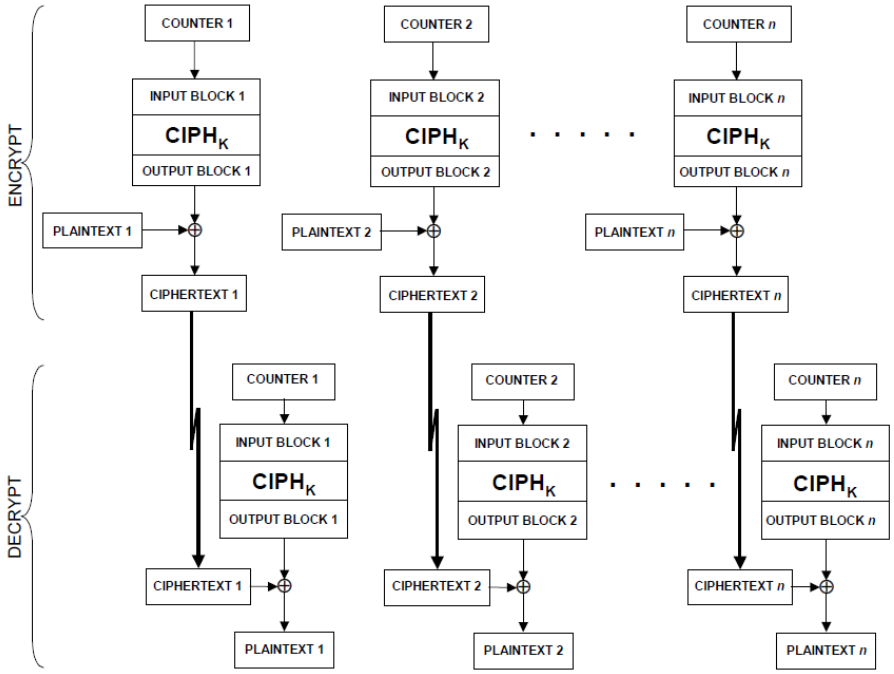

Figura 2.9: Counter Mode[4]

Para o ultimo bloco, que pode ser um bloco parcial de n bits, os n bits mais significativos do ultimo bloco de saída são usados para a operação de XOR, os restantes bits do bloco de saída são descartados.

Tanto na cifragem como na decifragem em modo *counter* o bloco pode ser executado em paralelo. Similarmente, o texto que corresponde a um bloco cifrado em particular pode ser recuperado independentemente de qualquer outro bloco se o correspondente *IV* possa ser determinado. A cifra pode ser aplicada aos *counters* em avanço, antes de os dados a serem cifrados ou decifrados estejam disponíveis.Dworkin (2001)Protection (2013)

Capítulo 3

Proposta de Trabalho

3.1 Implementação

Tipicamente a implementação das S-boxes para uma alta taxa de transferência é feita mediante a utilização de LUT onde o valor de substituição se encontra pre-computado, permitindo desta forma uma taxa de transferência elevada ao permitir que esta transformada possa ser executada em apenas um ciclo de relógio recorrendo a paralelização. Mas esta implementação consome muita memória ao ser necessária a replicação da LUT para cada byte e a replicação também de cada round, sendo necessárias 160 LUTS, 16 por byte mais 10 rounds, o que torna a sua implementação em ASIC proibitiva. Para se conseguir um baixo consumo de área, tem de se executar o calculo da S-box durante a execução, mas o calculo durante a execução da S-box usando GF(2^8) é complexa, reduzindo significativamente a taxa de transferência da cifra. Um dos inventores da cifra AES, sugeriu em Rijmen (2001) em vez do uso de GF(2^8) para a computação da s-box, a decomposição e respectivo calculo em GF(2^4). Cada elemento de GF(2^8) pode ser escrito como um polinomio de primeiro grau com coeficientes de GF(2^4).

O valor de entrada é mapeado em dois elementos GF(2^4), a seguir é calculada a multiplicativa inversa usando GF(2^4), sendo depois inversamente mapeados para GF(2^8). Esta implementação apresenta duas vantagens, a redução da complexidade dos cálculos executados, assim como a possibilidade de *sub-pipelining* podendo dessa forma fazer o calculo da s-box durante a execução, mantendo uma taxa de tranferencia próxima dos valores alcançados pela implementação através de *LUT*.

Embora a aplicação do calculo da S-box *"on-the-fly"* sobre GF(2^4), reduzir bastante a área utilizada, este também reduz significativamente a taxa de transmissão, devido a esta implementação sofrer de um caminho critico longo. Para ultrapassar esta desvantagem é usado *sub-pipelining*, dessa forma conseguiram atingir uma taxa de transferência próxima do conseguido através de *LUT*

Em Alireza Hodjat (2006) foram realizadas varias implementações da cifra AES, explorando o *trade-off* entre a área e a taxa de transmissão para implementação ASIC em tecnologia CMOS de $0.18\mu m$, tendo alcançado entre 30 a 70 Gbits/s dependo da implementação realizada. Foram utilizadas diferentes estrategias de implementação, tais como o uso de LUTS para o calculo das

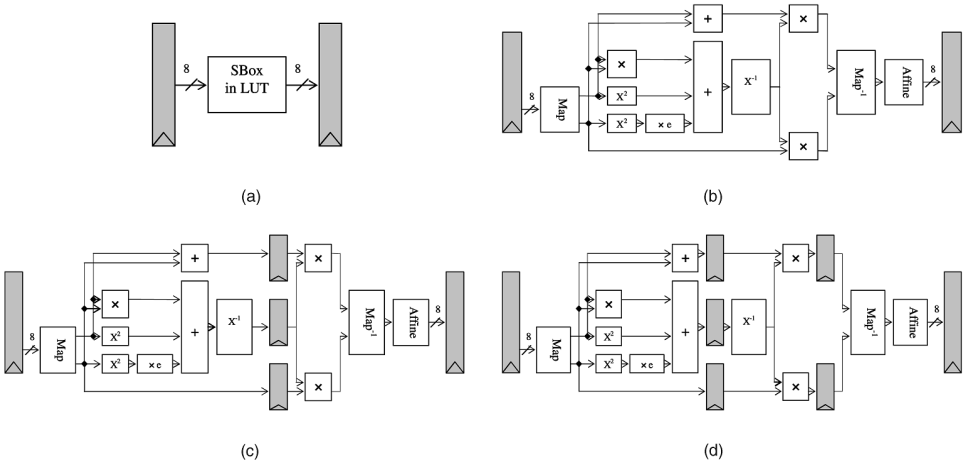

Figura 3.1: Implementações S-box realizadas em Alireza Hodjat (2006)

S-boxes ou executando o calculo das S-boxes *"on-the-fly"* em GF(2^4) utilizando diferentes estágios de *sub-pipelining*, assim como o calculo das sub-chaves pela tranformação key expansion *offline* e *online*. As varias implementações realizadas da transformação SubBytes são apresentadas na figura 3.1, onde a) corresponde a implementação usando LUT, b) implementação sem *sub-pipelining*, c) dois estágios de *sub-pipelining* e d) três estágios de *sub-pipelining*. Realizaram a comparação das varias implementações em relação a área utilizada e taxa de transferência máxima conseguida.

Na figura 3.2 são apresentados os resultados para o calculo das sub-chaves *online* e na figura 3.3 os resultados para o calculo das sub-chaves *offline*.

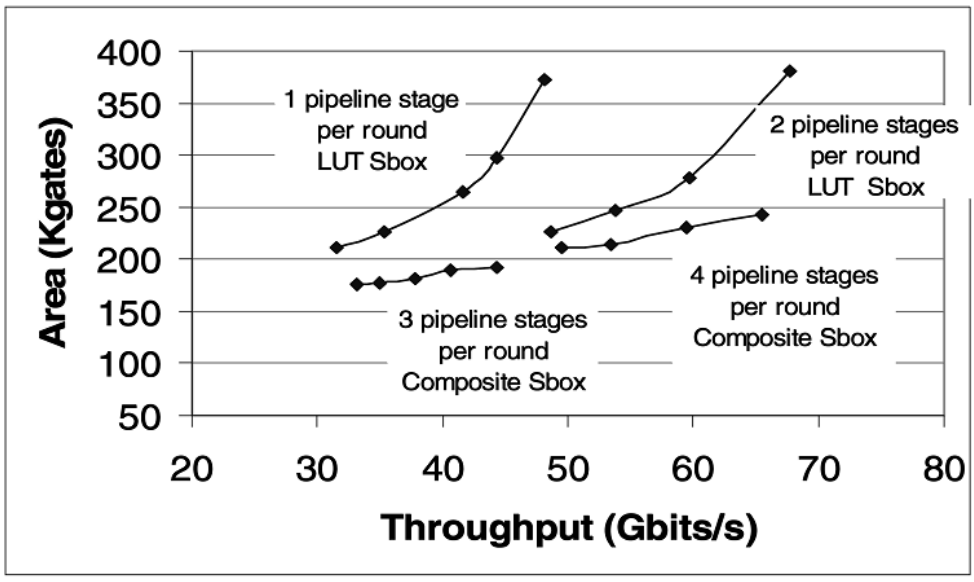

Figura 3.2: Area vs Taxa de transferencia com Key Expansion *online*

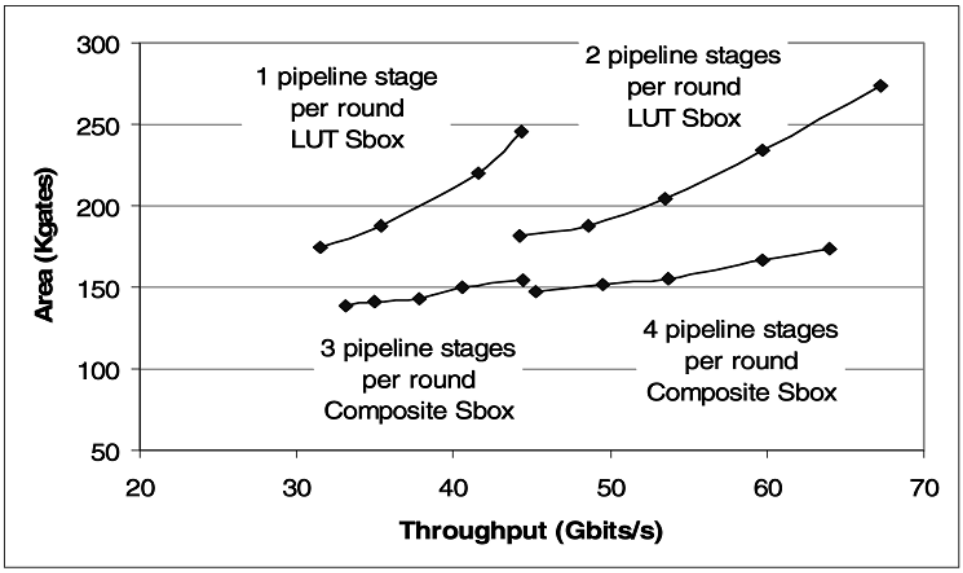

Figura 3.3: Area vs Taxa de transferencia com Key Expansion *offline*

Chegaram a conclusão de que, usando 3 estágios de *pipelining* dentro da transformação SubBytes conseguem diminuir a area em 32%. Concluíram também que como a chave de encriptação se mantêm constante durante uma sessão, ou seja, as sub-chaves de cada round são constantes durante uma sessão inteira, ao efetuar o calculo das sub-chaves para cada round no modo offline é conseguida dessa forma uma redução da area utilizada em 28%. Com a combinação desses dois métodos de implementação foi uma redução de 48% em relação a uma implementação com a mesma taxa de transferência utilizando *LUT* executando calculando da Key Expansion *online*.

Chethan Ananth (2001) propõem ainda a decomposição de GF(2^4) em GF(2^2) e as operações GF(2^2) em GF(2), podendo desta forma toda a implementação S-box ser feita usando simplesmente bit XOR para adição e bit AND para a multiplicação. A inversão em GF(2) do valor 1 é 1 e a inversão de 0 não existe. Por isso toda a inversão em GF(2^8) pode ser decomposta em circuitos lógicos usando unicamente gate XORS e gates AND. O principal objectivo deste artigo é obter um grande taxa de transferência, tendo para isso recorrido a uma implementação *full pipelined* . Foi feito uma implementação com *pipeline* balanceado, em que foi calculado o *delay* de cada operação basica, e foram aplicados os registros de *pipelining* de modo a balancear o tempo entre cada registro de pipeline. A implementação foi realizada usando uma FPGA Virtex 5. Os resultados são apresentados na figura 3.2.

3.2 Modelo de Alto Nível

Foi implementado o modelo de alto nível não sintetizável de modo a fazer a verificação funcional da cifra AES, esta foi implementada usando loop das várias rounds e a transformação Key Expansion foi implementada usando o modo *offline*, em que as sub-chaves foram todas calculadas

Pipeline stages	Throughput (Gbps)	Area (slice)	BRAM (%)	Latency (ns)
1	20	8,800	0	6.25
2	27	9,086	0	4.67
4	48	10,561	0	2.777
8	67	13,649	0	1.905

Figura 3.4: Resultados obtidos por Hodjat e Verbauwhede (2004)

de uma só vez e guardadas num bloco de memória, sendo depois acedida em cada AddRoundKey. A implementação da transformação SubBytes foi feita com recurso a LUT, visto que o modelo serve apenas para verificar o funcionamento da cifra e este método de implementação é o mais simples, que será posteriormente alterado para outro mais ajustado a uma implementação ASIC.

3.2.1 SubBytes

Esta foi implementada com recurso a *LUT*, visto que o modelo serve apenas para verificar o funcionamento da cifra e este método de implementação é o mais simples, que será posteriormente alterado para outro mais ajustado a uma implementação ASIC.

```
case (din)          //Look Up Table
  8'h00              : dout <= 8'h63;
  8'h01              : dout <= 8'h7c;
  8'h02              : dout <= 8'h77;
  8'h03              : dout <= 8'h7b;
  8'h04              : dout <= 8'hf2;
  8'h05              : dout <= 8'h6b;
  8'h06              : dout <= 8'h6f;
  8'h07              : dout <= 8'hc5;
  8'h08              : dout <= 8'h30;
  8'h09              : dout <= 8'h01;
```

Nesta implementação, construi-se um ciclo case, em que foram mapeadas todas as entradas possíveis, sendo din o valor do byte a ser substituído.

3.2.2 Key Expansion

Na implementação da Key Expansion, foi feita a computação da mesma no modo *offline*, onde todas as sub-chaves são calculadas de uma so vez e guardadas em memória

```
for(i=1;i<=10;i=i+1)begin

for(j=1;j<=3;j=j+1) begin
assign W[(4*i)+j] = W[(4*i)+j-1] ^ W[4*(i-1)+j];
end

SBox XGY(clk,reset,valid_in,W[(4*i)-1][31:24],V[(4*(i-1))]);
SBox XGA(clk,reset,valid_in,W[(4*i)-1][23:16],V[(4*(i-1)+1)]);
SBox XGB(clk,reset,valid_in,W[(4*i)-1][15:8],V[(4*(i-1)+2)]);
SBox XGC(clk,reset,valid_in,W[(4*i)-1][7:0],V[(4*(i-1)+3)]);

assign AUX[i] = {(V[(4*(i-1)+1)]^RCON[i-1]),
V[(4*(i-1)+2)],
V[(4*(i-1)+3)],
V[(4*(i-1))]};
assign W[4*i] = W[4*(i-1)] ^ AUX[i];
end
```

No ciclo for encadeado $for(j = 1; j <= 3; j = j + 1)$ são calculadas todas as Words excepto a mais significativa que formam cada sub-chave.

Para calcular a Word mais significativa, esta é composta por a Word menos significativa pertencente a mesma sub-chave após sofrer uma transformação g() Xored com a Word mais significativa da sub-chave anterior.

```
 SBox XGY(clk,reset,valid_in,W[(4*i)-1][31:24],V[(4*(i-1))]);
 SBox XGA(clk,reset,valid_in,W[(4*i)-1][23:16],V[(4*(i-1)+1)]);
 SBox XGB(clk,reset,valid_in,W[(4*i)-1][15:8],V[(4*(i-1)+2)]);
 SBox XGC(clk,reset,valid_in,W[(4*i)-1][7:0],V[(4*(i-1)+3)]);

 assign AUX[i] = {(V[(4*(i-1)+1)]^RCON[i-1]),
 V[(4*(i-1)+2)],
 V[(4*(i-1)+3)],
 V[(4*(i-1))]};
```

Esta porção de código é a transformação g(), onde primeiramente a Word menos significativa é aplicada a transformação SubBytes, sendo depois efecuada uma rotação de um byte a esquerda e é feita um Xor com a constante RCON[i], o valor é guardado na variavel auxiliar AUX[i] e depois é feito um xor com a Word mais significativa da sub-chave anterior, formando a Word mais significativa da sub-chave a ser calculada no momento.

3.2.3 MixColumns

No código em baixo, é feito o preenchimento da matriz de estado, data_in é uma variavel de 128 bits.

```
for(i=0; i<=15; i=i+1) begin
 assign state_aux[i]= data_in[(((15-i)*8)+7):((15-i)*8)];
end
```

a multiplicação é feita através de uma rotação a esquerda de um bit, a não ser que o ultimo bit seja um, nesse caso é feito uma rotação a esquerda de um bit seguida de uma subtração com o polinômio irredutível.

```
for(i=0; i<=15; i=i+1) begin
assign state_mul[i]=(state_aux[i][7])?((state_aux[i]<<1)^8'h1b):(state_aux[i]<<1);
```

Aqui são aplicadas as equações (2.1) do Capitulo 2. Em que a multiplicação pelo valor 3 corresponde a multiplicação por dois mais a adição do próprio valor através de um XOR.

```
state[0]= state_mul[0]^(state_mul[1]^state_aux[1])^state_aux[2] ^state_aux[3];
```

3.2.4 ShifRows

Tal como na implementação Mixcolumns, os dados de entrada vêm num *array* de 128 bits, e são dispostos na matriz de estado. Sendo depois rearranjados, como demonstra o pedaço de código dem baixo de modo a que a segunda linha seja rodada de um byte a esquerda, a terceira de dois, e a quarta de três como explicado no Capitulo 2. Sendo depois os bytes da matrix de estado concatenados para o array de saida de 128 bits.

```
data_out[(15*8)+7:(12*8)]<={state[0],state[5],state[10],state[15]};
data_out[(11*8)+7:(8*8)]<={state[4],state[9],state[14],state[3]};
data_out[(7*8)+7:(4*8)]<={state[8],state[13],state[2],state[7]};
data_out[(3*8)+7:(0*8)]<={state[12],state[1],state[6],state[11]};
```

3.2.5 AddRoundKey

Corresponde simplesmente numa leitura da sub-chave da memoria onde foram guardadas todas as sub-chaves na transformação Key Expansion e um XOR entre os 128 bits da matriz de estado com a sub-chave de cada ronda.

3.3 Plano de trabalho

As etapas identificadas até ao momento e suas respetivas durações são:

- Aprendizagem das ferramentas a usar durante a execução do projeto (1 semana);
- Desenvolvimento detalhado da arquitectura (4 semanas).
- Implementação do Código em Verilog(5 semanas).
- Análise, comparação e optimização de resultados (5 semanas).
- Redação da dissertação (4 semanas).

O Diagrama de Gantt associado ao plano de trabalho pode ser consultado na figura 3.5.

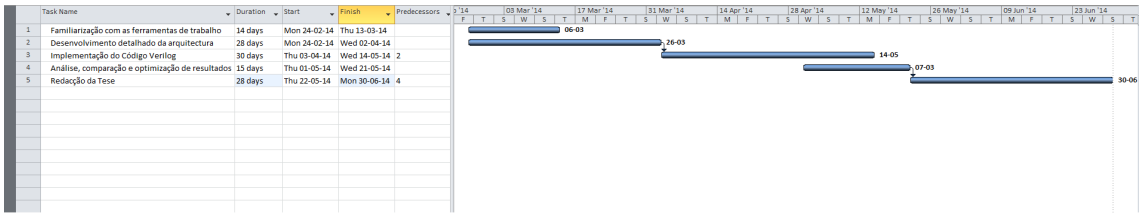

Figura 3.5: Diagrama de Gantt

Referências

Ingrid Verbauwhede Alireza Hodjat. Area-throughput trade-offs for fully pipelined 30 to 70 gbits/s aes processors. *IEEE TRANSACTIONS ON COMPUTERS, VOL. 55, NO.4*, 2006.

PAGES=87-117 BOOKTITLE=Understanding-Cryptography PUBLISHER=Springer-Verlag MONTH= YEAR=2010 C. Paar and J. Pelzl, TITLE=The Advanced Encryption Standard(AES).

Karthick Ramu Chethan Ananth. Fully pipelined implementations of aes with speeds exceeding 20 gbits/s with s-boxes implemented using logic only. *Proceedings of the 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, 2001.

Morris Dworkin. Recommendation for block cipher modes of operation. Relatório técnico, National Institute os Standards and Technology(NIST), December 2001.

Alireza Hodjat e Ingrid Verbauwhede. A 21.54 gbits/s fully pipelined aes processor on fpga. *Proceedings of the 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, 2004.

IDC. Hdmi: The digital display link, December 2006. URL http://www.hdmi.org/pdf/whitepaper/SilicaonImageHDMIWhitePaperv73(2).pdf.

DIGITAL CONTENT PROTECTION. Digital content protection for new home theater networking scenarios, November 2008. URL http://www.digital-cp.com/files/documents/A1DEB6A1-E35C-E3D1-9B4327AECFDB6B3E/DCP_NHTNWP[110408][v02].pdf.

Digital Content Protection. Mapping hdcp to hdmi revision 2.2. Relatório técnico, High-bandwidth Digital Content Protection System, Ferbuary 2013.

FIPS PUBS. Advanced encryption standard (aes). Relatório técnico, Federal Information Preocessing Standards Publications, November 2001.

Vincent Rijmen. Efficient implementation of the rijndael s-box. *Proceedings of the 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, 2001.

Synopsys. Corporate backgrounder, 2014. URL http://www.synopsys.com/Company/AboutSynopsys/Pages/CompanyProfile.aspx.