SD192 – Trabalho Orientado I

Implementação em Verilog de Criptografia AES-128

com Protocolo de Comunicação I²C

Execução
Inatel UNIFEI Instituto Hardware BR Uema cepedi DIGITAL
Realização
CHIP TECH Softex Ministério da Ciência, Tecnologia e Inovação Governo Federal

Autores
André Luiz E. Araújo
Daniella Vicentini Azevedo dos Santos
Guilherme Henrique Duarte Mendes
Janaína da Glória Moreira de Oliveira
Lucas Manoel Leite de Souza
Maria Tereza Rocha Carvalho
Matheus Henrique Martins Paiva
Sérgio Henrique Azevedo dos Santos

Orientador
Felipe Gustavo de Freitas Rocha

TÓPICOS

- Introdução
- Arquitetura Proposta e testes
- Conclusão

Implementação completa do algoritmo AES-128 em Verilog:

- **Conceito sobre AES-128**
- **Integração com interface I^2C**
- **Blocos para criptografia e descriptografia**
- **Módulo TOP controlador**
- **Validação com vetores padrão FIPS-197**

INTRODUÇÃO

CONTEXTO E MOTIVAÇÕES

Segurança em Hardware

Implementações em Hardware oferecem maior proteção contra ataques de software e side-channel, essenciais para sistemas críticos.

Desempenho Superior

Criptografia em Hardware é amplamente superior em performance se comparada a implementações em software, crucial para aplicações em tempo real.

Eficiência Energética:

Circuitos dedicados consomem significativamente menos energia, ideal para dispositivos IoT e embarcados com restrições de energia.

Gerenciamento Seguro de Chaves:

A interface I2C permite atualização remota e segura das chaves de criptografia, mantendo a flexibilidade do sistema.

O AES-128

Advanced Encryption Standard:

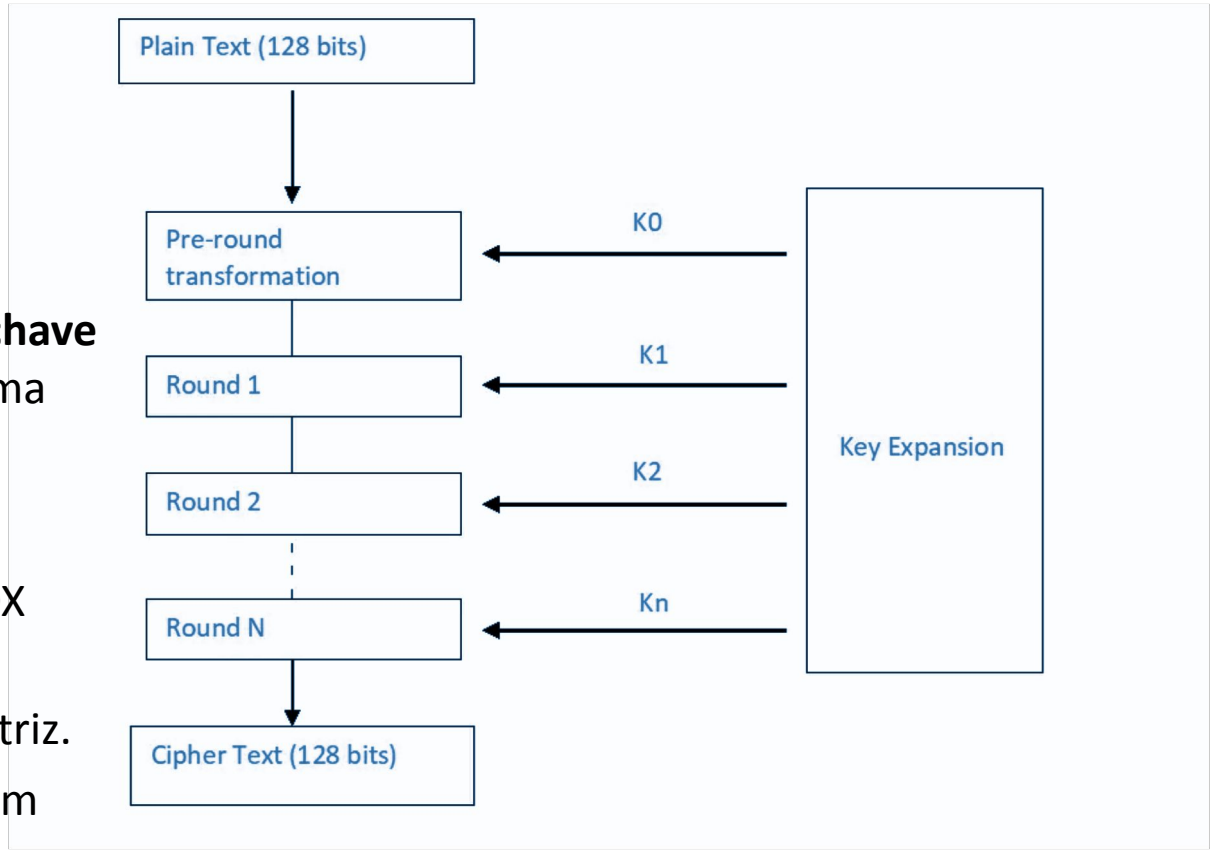
- Padronizado pelo NIST em 2001
- Substituiu o DES (maior segurança, eficiência e confiabilidade).
- Seguro até para dados **TOP SECRET**.

Estrutura Operacional:

AES-128 processa blocos de **128 bits** (16 bytes) com uma **chave de 128 bits**, distribuídas ao longo de **10 rodadas**. Utiliza uma **matriz 4x4 (**cada elemento representa 1 byte da palavra).

Etapas por Rodada:

- **SubBytes:** substituição de cada byte via tabela SBOX (não-linear).
- **ShiftRows:** deslocamento circular das linhas da matriz.
- **MixColumns:** multiplicação matricial de colunas com operações em GF(2^8).
- **AddRoundKey:** operação XOR com subchave da rodada.



RODADAS DO AES-128

- **1 Pré-Rodada**
- **9 Rodadas Completas (1 a 9)**
- **1 Rodada Final (10ª Rodada)**

Cada rodada transforma a **matriz de estado 4x4** (representando os 128 bits de dados) por meio de operações criptográficas.

Pré Rodada (Rodada 0)

- Operação: ADD Round Key
- Descrição: O estado inicial (plaintext de 128 bits) é **combinado com a chave original** por meio da operação XOR.
- Objetivo: Inserir a Chave no sistema antes de qualquer transformação.

RODADAS DO AES-128

Rodadas Completas de 1 a 9 (quatro transformações básicas):

1. **SubBytes**

   Cada byte da matriz de estado é substituído por um valor da **SBOX** (tabela padrão de 256 entradas criada com base em transformações inversas em campos finitos e operações afins). Essa substituição introduz **não-linearidade** ao sistema, tornando-o resistente a ataques lineares e diferenciais.

2. **ShiftRow**

   Desloca ciclicamente os bytes das linhas 1, 2 e 3 da matriz (a linha 0 permanece).

- Linha 0: sem deslocamento
- Linha 1: 1 byte à esquerda
- Linha 2: 2 bytes
- Linha 3: 3 bytes

Esse deslocamento reorganiza os dados horizontalmente, espalhando os bytes em colunas diferentes.

RODADAS DO AES-128

3. **MixColumns**
   Cada coluna da matriz é multiplicada por uma **matriz constante 4x4 bytes no campo finito GF(2^8)** (representação polinomial de grau 7). Essa operação mistura os bytes de cada coluna por meio de multiplicações e somas modulares, fazendo com que a alteração de um único byte afete todos os bytes da coluna.

4. **AddRoundKey**

   A matriz de estado é combinada com uma subchave de 128 bits gerada pela expansão da chave original através da **XOR bit a bit**. É a única etapa que efetivamente insere a chave secreta no processo.

Rodadas do AES-128

Rodada 10 – Rodada final

Executa apenas **3 das 4 operações**:

1. SubBytes
2. ShiftRows
3. AddRoundKey

MixColumns é omitido propositalmente.

Por quê?
A omissão do MixColumns na última rodada garante a simetria, diminui a complexidade da implementação e o tempo de execução, mantendo o equilíbrio entre segurança e possibilidade de recuperação durante a descriptografia, definido pela norma FIPS-197.

A ordem das operações é determinística e essencial. Alterar a ordem ou omitir qualquer uma das etapas compromete totalmente a segurança e o funcionamento do algoritmo.

I2C

1. Inter-Integrated Circuit (I2C):

. Protocolo de comunicação serial desenvolvido pela Philips, que utiliza apenas dois fios (SDA e SCL) para conectar múltiplos dispositivos.

2. Arquitetura Mestre-Escravo:

. Um dispositivo mestre inicia e controla a comunicação, enquanto os dispositivos escravos (slaves) respondem aos comandos do mestre.

3. Funcionamento do Slave:

. Cada slave possui um endereço único. O slave monitora o barramento, reconhece seu endereço e responde apenas quando solicitado pelo mestre.

4. Aplicação no Projeto:

. O módulo I2C Slave recebe a chave de criptografia de 128 bits enviada por um dispositivo mestre externo, permitindo a atualização segura da chave sem redesenhar o hardware.

MOTIVOS PARA ADOTAR I2C SLAVE

1. Função reativa do circuito AES

- O módulo de criptografia não tem iniciativa de comunicação. Ele apenas aguarda dados de entrada (chave, palavra e comando) e processa conforme solicitado. Sendo assim, faz mais sentido que o dispositivo externo (por exemplo, um microcontrolador) atue como Master, enquanto nosso módulo se comporta como um periférico receptivo (Slave).

2. Compatibilidade com Sistemas Externos

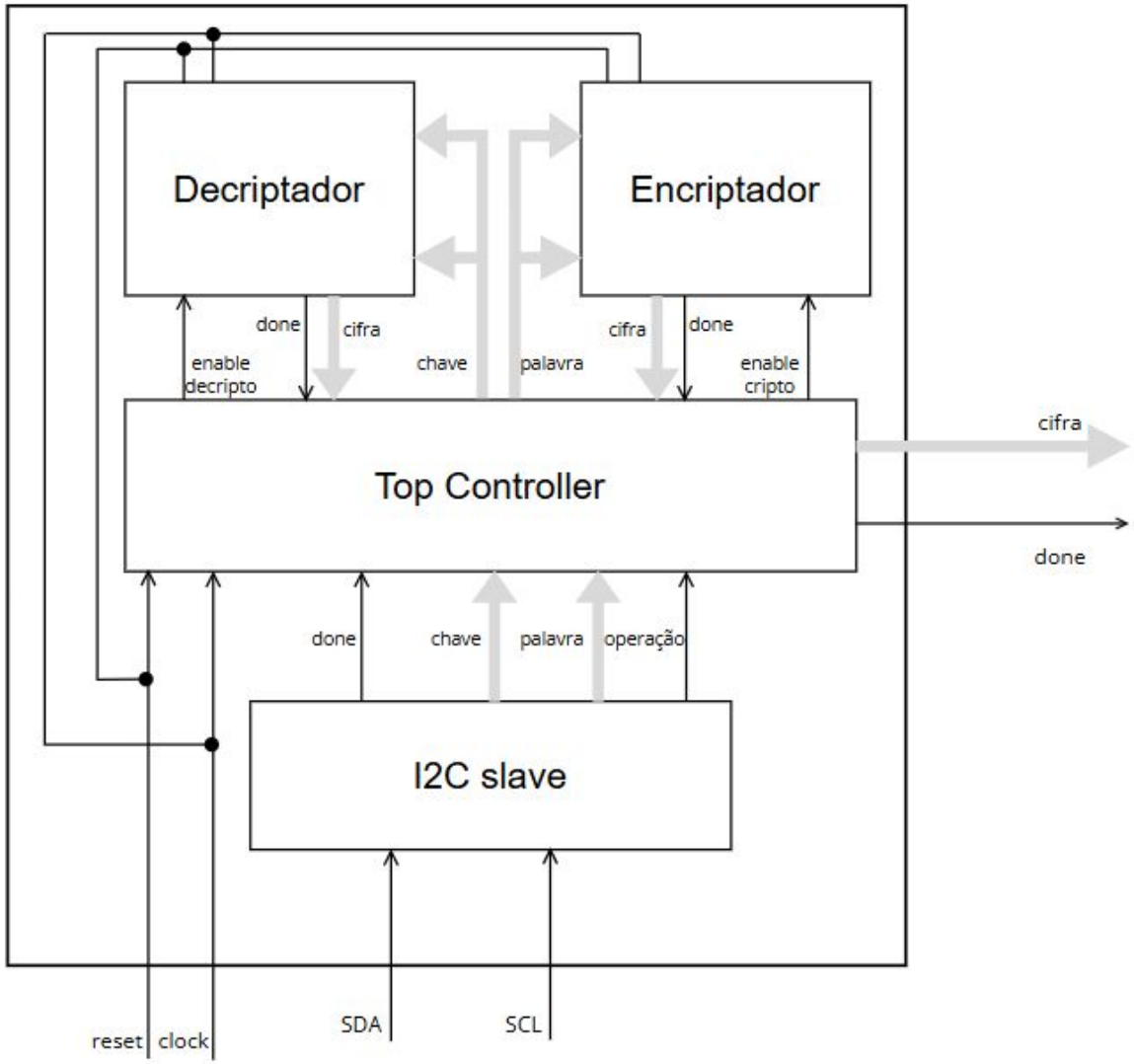
- A maioria dos controladores ou SoCs comerciais já operam como I^2C Masters por padrão. Ao projetarmos o nosso sistema como Slave, garantimos fácil integração com sistemas reais, como Raspberry Pi, STM32, ESP32, etc.

3. Simplicidade na atualização de dados

- O modo Slave permite que a chave e a palavra sejam atualizadas dinamicamente via comunicação serial, sem necessidade de reconfiguração física ou redesenho do hardware. Isso garante flexibilidade e escalabilidade.

ARQUITETURA PROPOSTA E TESTES

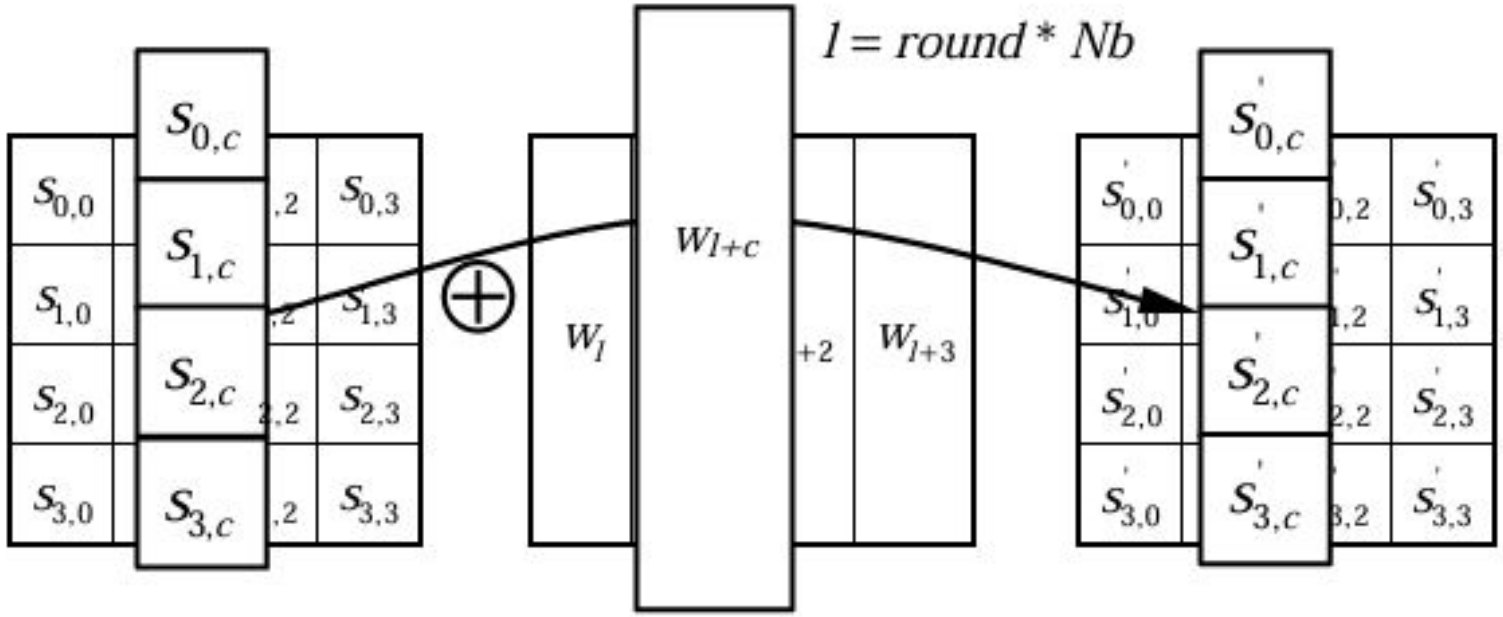
ARQUITETURA



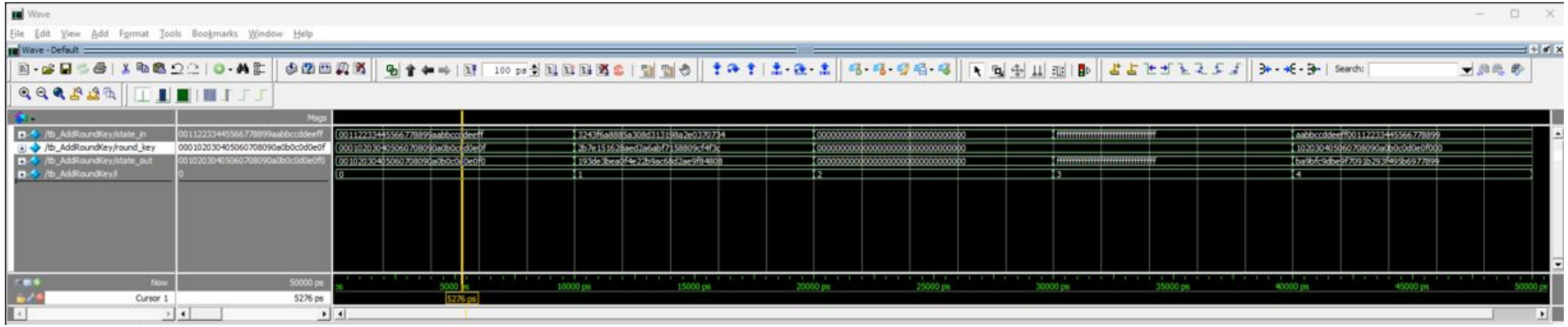
ADDROUNDKEY

Um módulo combinacional que realiza uma operação XOR entre o estado atual e a subchave da rodada

- Sem uso de clock
- Opera diretamente sobre os 128 bits da matriz de estado
- Usa a lógica XOR bit a bit com a subchave da rodada.



TB_ADDROUNDKEY




```
Teste 0:
  State In    : 00112233445566778899aabbccddeeff
  Round Key   : 000102030405060708090a0b0c0d0e0f
  Resultado   : 00102030405060708090a0b0c0d0e0f0
  Esperado    : 00102030405060708090a0b0c0d0e0f0
  --> OK

Teste 1:
  State In    : 3243f6a8885a308d313198a2e0370734
  Round Key   : 2b7e151628aed2a6abf7158809cf4f3c
  Resultado   : 193de3bea0f4e22b9ac68d2ae9f84808
  Esperado    : 193de3bea0f4e22b9ac68d2ae9f84808
  --> OK

Teste 2:
  State In    : 00000000000000000000000000000000
  Round Key   : 00000000000000000000000000000000
  Resultado   : 00000000000000000000000000000000
  Esperado    : 00000000000000000000000000000000
  --> OK
```

```
Teste 3:
  State In    : ffffffffffffffffffffffffffffffff
  Round Key   : 00000000000000000000000000000000
  Resultado   : ffffffffffffffffffffffffffffffff
  Esperado    : ffffffffffffffffffffffffffffffff
  --> OK

Teste 4:
  State In    : aabbccddeeff00112233445566778899
  Round Key   : 102030405060708090a0b0c0d0e0f000
  Resultado   : ba9bfc9dbe9f7091b293f495b6977899
  Esperado    : ba9bfc9dbe9f7091b293f495b6977899
```

S-BOX

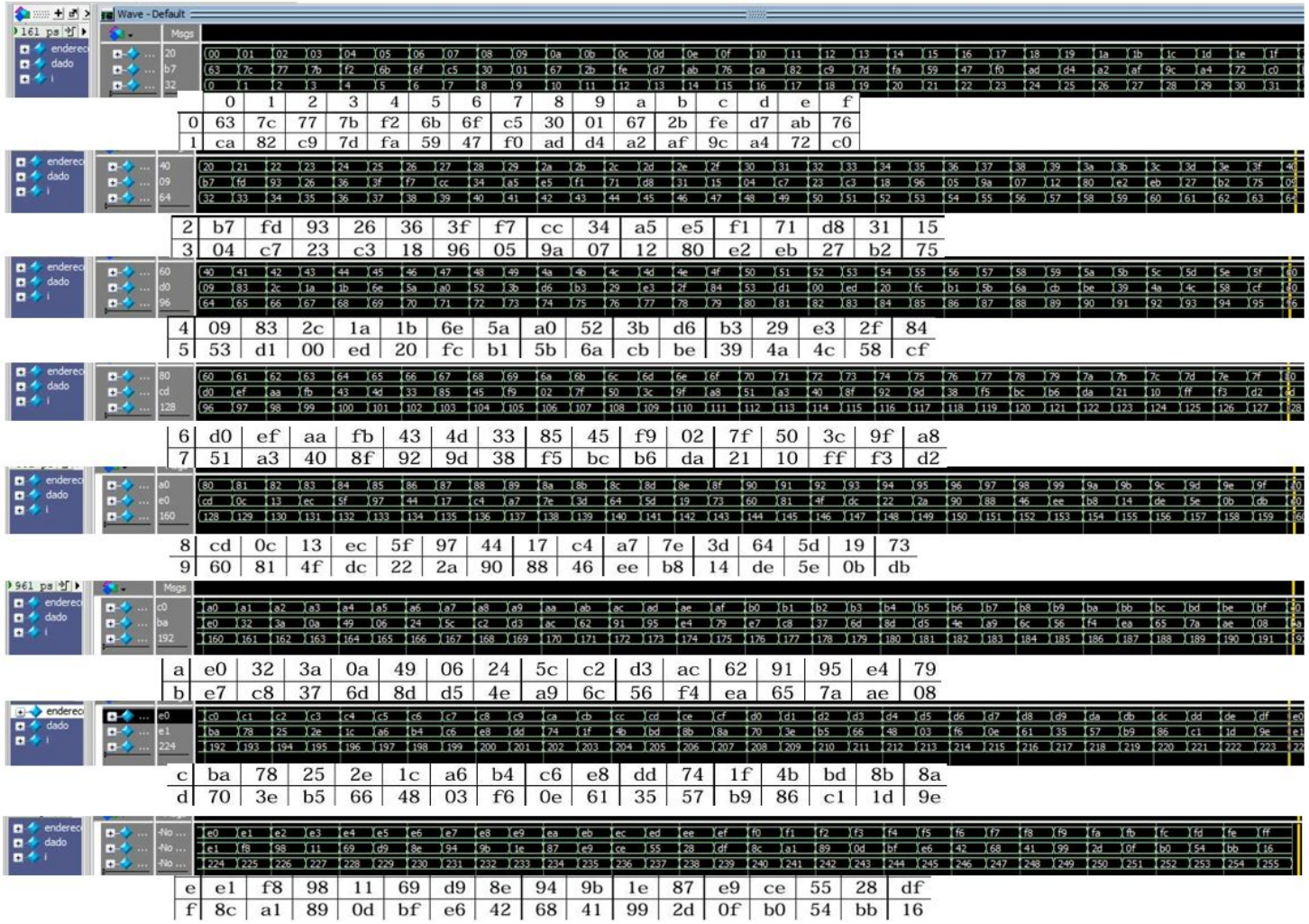
- Case com os 256 valores possíveis para 2 bytes. Valores fornecidos na FIPS 197.

Entradas: Valor procurado - 8 bits
Saída: Valor a substituir - 8 bits

		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

TB_SBOX

Módulo utilizando *for* para testar os 256 valores possíveis da tabela.



	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
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

INV_SBOX

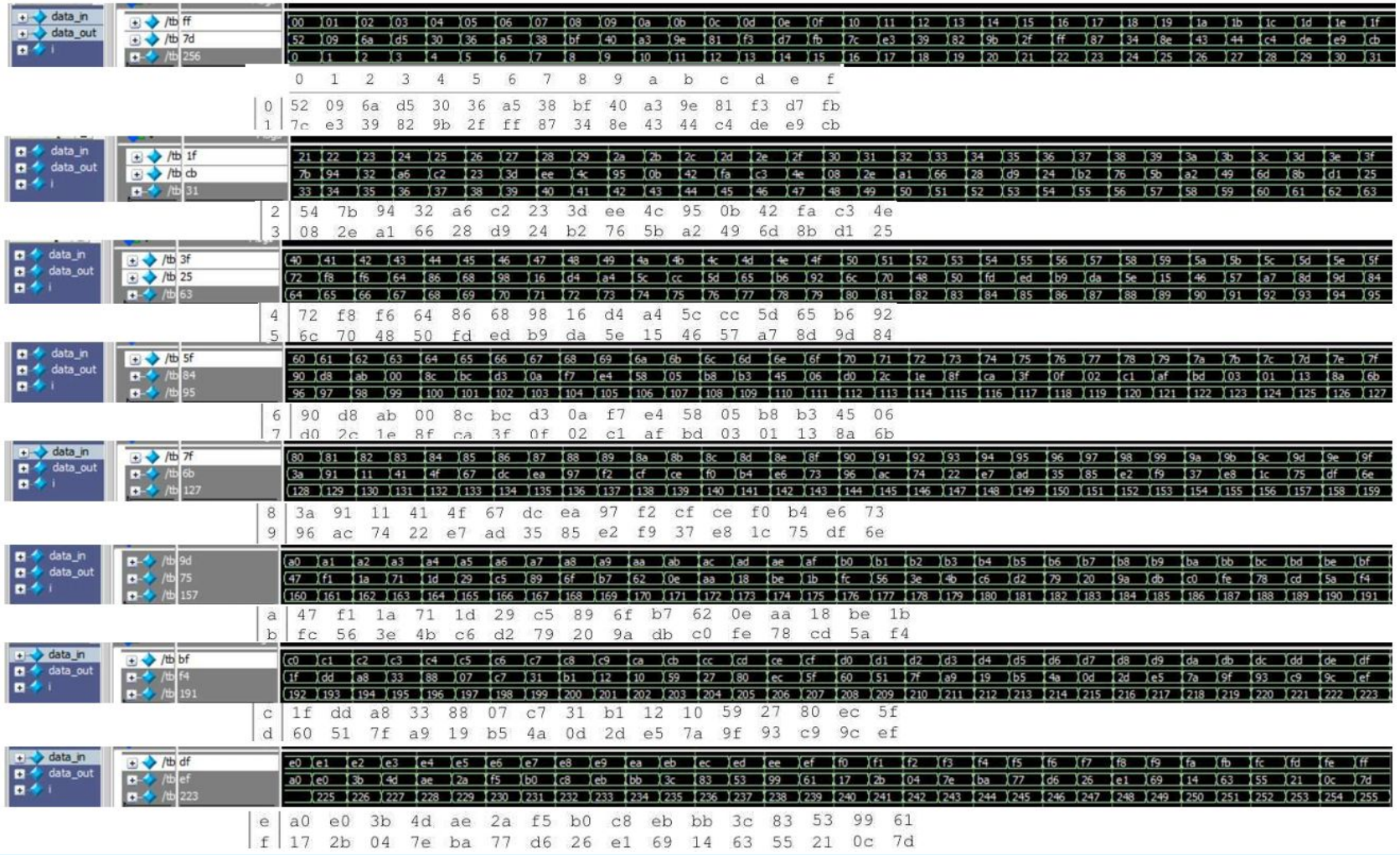
Módulo responsável pela substituição de um byte por outro, seguindo uma tabela padrão InvS-Box.

- data_in - Um byte que será o valor a ser substituído
- data_out - Um byte que será o valor correspondente da tabela S-Box.

		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
	1	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
	2	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
	3	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
	4	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
	5	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
	6	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
	7	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
	8	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
	9	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
	a	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
	b	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
	c	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
	d	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
	e	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
	f	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

TB_INV_SBOX

Módulo utilizando *for* para testar os 256 valores possíveis da tabela.

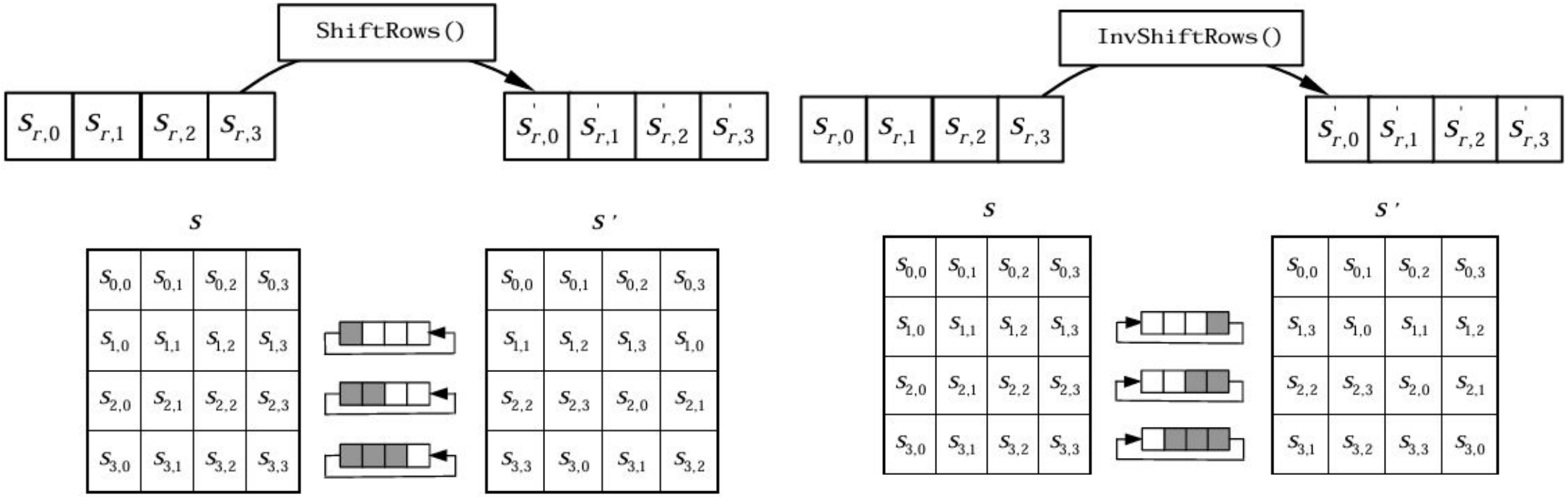


	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
1	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
2	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
3	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
4	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
5	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
6	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
7	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
8	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
9	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
a	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
b	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
c	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
d	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
e	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
f	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

SHIFTROWS E INV_SHIFTROWS

Módulos que implementam a transposição de linhas da matriz de estado, aumentando a difusão dos dados.

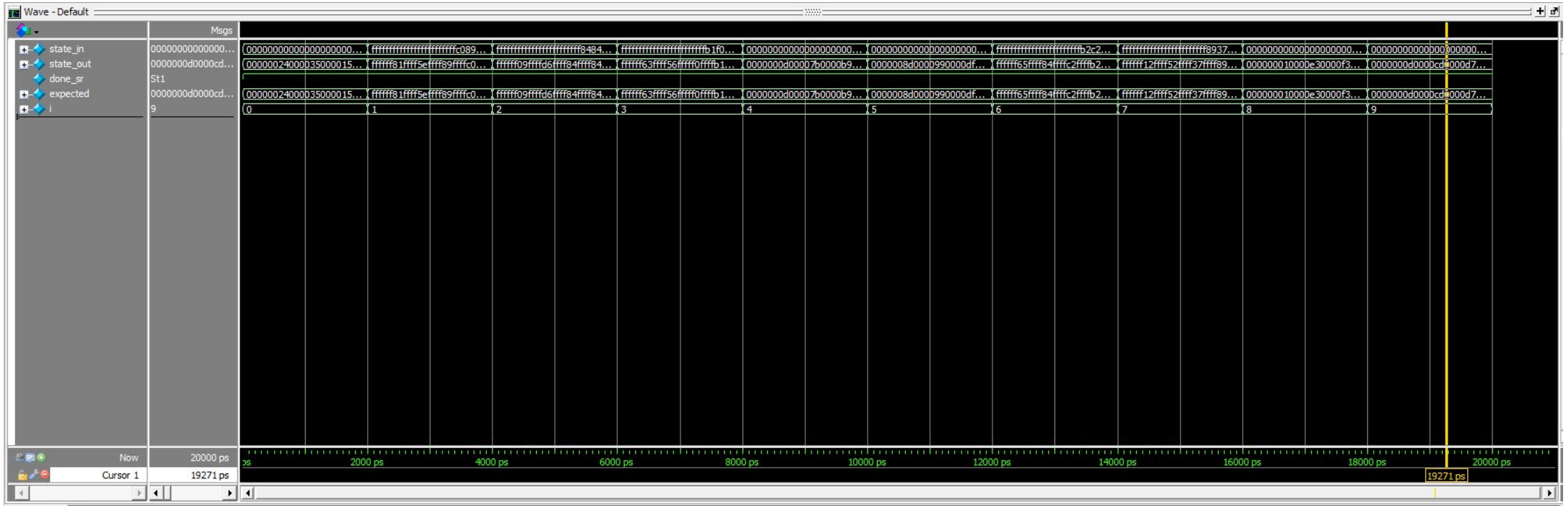
- **ShiftRows:** desloca linhas para a esquerda.
- **InvShiftRows:** desloca para a direita.
- Entrada: vetor de 128 bits representando a matriz de estado.
- Saída: vetor reordenado conforme FIPS-197.



TB_SHIFTROWS E TB_INV_SHIFTROWS

```
VSIM 7> run -all
# Teste 0
# state_in  = 00000000000000000000000012153524
# esperado = 00000024000035000015000012000000
# obtido   = 00000024000035000015000012000000
# done_sr     = 1
# Sucesso: saida correta e done_sr ativo.
#
# Teste 1
# state_in  = ffffffffffffffffffffffc0895e81
# esperado = ffffff81ffff5efffff89ffffc0ffffff
# obtido   = ffffff81ffff5efffff89ffffc0ffffff
# done_sr     = 1
# Sucesso: saida correta e done_sr ativo.
#
# Teste 2
# state_in  = ffffffffffffffffffffffff8484d609
# esperado = ffffff09ffffd6ffff84ffff84ffffff
# obtido   = ffffff09ffffd6ffff84ffff84ffffff
# done_sr     = 1
# Sucesso: saida correta e done_sr ativo.
#
# Teste 3
# state_in  = ffffffffffffffffffffffffb1f05663
# esperado = ffffff63ffff56ffffff0ffffb1ffffff
# obtido   = ffffff63ffff56ffffff0ffffb1ffffff
# done_sr     = 1
# Sucesso: saida correta e done_sr ativo.
#
# Teste 4
# state_in  = 0000000000000000000000006b97b0d
# esperado = 0000000d00007b0000b9000006000000
# obtido   = 0000000d00007b0000b9000006000000
# done_sr     = 1
# Sucesso: saida correta e done_sr ativo.
#
# Teste 5
# state_in  = 0000000000000000000000046df998d
# esperado = 0000008d0000990000df000046000000
# obtido   = 0000008d0000990000df000046000000
# done_sr     = 1
# Sucesso: saida correta e done_sr ativo.
#
```

TB_SHIFTROWS E TB_INV_SHIFTROWS



MIXCOLUMNS E INV_MIXCOLUMNS

MixColumns: Mistura os bytes de cada coluna da matriz de estado.

- Multiplica a matriz de estado 4×4 por uma matriz fixa
- A saída de cada byte passa a depender de todos os bytes da mesma coluna.
- As operações ocorrem no Campo de Galois GF(2^8):
  - Soma: XOR bit a bit.
  - Multiplicação por 2: deslocamento à esquerda com correção por XOR com 0x1b se MSB = 1.
  - Multiplicação por 3: xtime(x) ^ x.

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

InvMixColumns: Utiliza a mesma arquitetura com a matriz inversa.

- Cada coeficiente é decomposto em multiplicações sucessivas por 2 (xtime), com combinações XOR.

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

TB_MIXCOLUMNS E TB_INV_MIXCOLUMNS

MixColumns:

```
# ===== Teste do mÃ³dulo MixColumns =====
# Entrada : 6353e08c0960e104cd70b751bacad0e7
# SaÃ-da   : 5f72641557f5bc92f7be3b291db9f91a
# Entrada : a7bela6997ad739bd8c9ca451f618b61
# SaÃ-da   : ff8796843ld86a5164515lfa773ad009
# Entrada : 3bd92268fc74fb735767cbe0c0590e2d
# SaÃ-da   : 4c9cle66f771f0762c3f868e534df256
# Entrada : 2d6d7ef03f33e334093602dd5bfbl2c7
# SaÃ-da   : 6385b79ffc538df997be478e7547d691
# Entrada : 36339d50f9b539269f2c092dc4406d23
# SaÃ-da   : f4bcd45432e554d075f1d6c51dd03b3c
#
# Entrada : e8dab6901477d4653ff7f5e2e747dd4f
# SaÃ-da   : 9816ee7400f87f556b2c049c8e5ad036
#
# Entrada : b458124c68b68a014b99f82e5f15554c
# SaÃ-da   : c57elcl59a9bd286f05f4be098c63439
#
# Entrada : 3elc22c0b6fcbf768da85067f6170495
# SaÃ-da   : baa03de7alf9b56ed5512cba5f414d23
#
# Entrada : 54d990al6ba09ab596bbf40eall1702f
# SaÃ-da   : e9f74eec023020f61bf2ccf2353c21c7
#
# Entrada : 00000000000000000000000000000000
# SaÃ-da   : 00000000000000000000000000000000
#
# Entrada : ffffffffffffffffffffffffffffffff
# SaÃ-da   : ffffffffffffffffffffffffffffffff
```

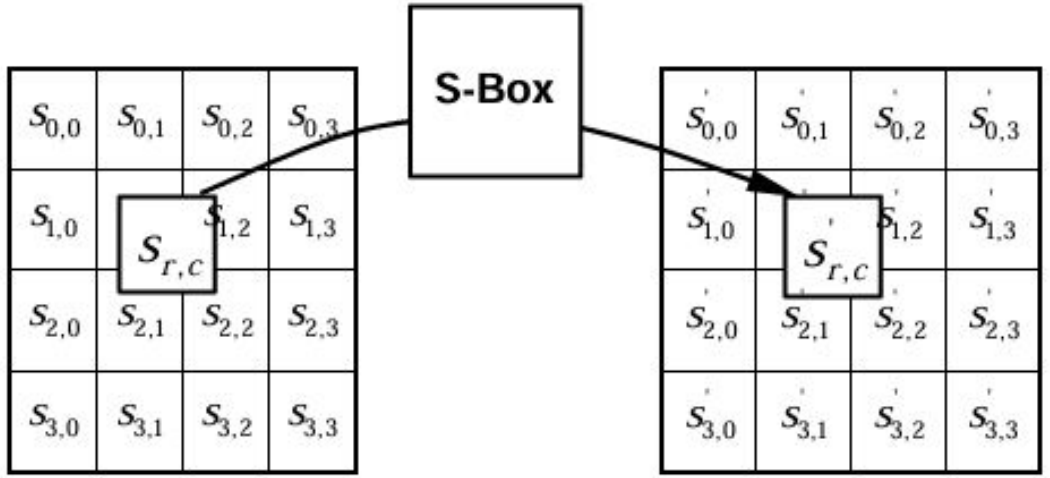
InverseMixColumns:

```
VSIM 13> run -all
# Entrada : bd6e7c3df2b5779e0b61216e8b10b689
# SaÃ-da   : 4773b91ff72f354361cb018eale6cf2c
# Entrada : fde3bad205e5d0d73547964ef1fe37f1
# SaÃ-da   : 2d7e86a339d9393ee6570a1101904e16
# Entrada : d1876c0f79c4300ab45594add66ff41f
# SaÃ-da   : 39daee38f4f1a82aaf432410c36d45b9
# Entrada : c62fe109f75eedc3cc79395d84f9cf5d
# SaÃ-da   : 9a39bf1d05b20a3a476a0bf79fe51184
# Entrada : c81677bc9b7ac93b25027992b0261996
# SaÃ-da   : 18f78d779a93eef4f6742967c47f5ffd
#
# Entrada : 247240236966b3fa6ed2753288425b6c
# SaÃ-da   : 85cf8bf472d124c10348f545329c0053
#
# Entrada : fa636a2825b339c940668a3157244d17
# SaÃ-da   : fc1fc1f91934c98210fbfb8da340eb21
#
# Entrada : 4915598f55e5d7a0daca94fa1f0a63f7
# SaÃ-da   : 076518f0b52ba2fb7a15c8d93be45e00
#
# Entrada : 89d810e8855ace682d1843d8cb128fe4
# SaÃ-da   : ef053f7c8b3d32fd4d2a64ad3c93071a
#
# Entrada : 00000000000000000000000000000000
# SaÃ-da   : 00000000000000000000000000000000
#
# Entrada : ffffffffffffffffffffffffffffffff
# SaÃ-da   : ffffffffffffffffffffffffffffffff
# ** Note: $stop    : C:/Users/regin/OneDrive/Ár
#    Time: 275 ps  Iteration: 0  Instance: /tb_i
```

SUBBYTE E INVSUBBYTE

SubBytes:

- Cada byte da matriz de estado é substituído usando a tabela S-box.
- O byte de entrada (8 bits) é tratado como índice da tabela:
  - 4 bits mais significativos → linha
  - 4 bits menos significativos → coluna
- A saída é o valor correspondente da S-box.



InvSubBytes:

- Mesma lógica estrutural.
- Utiliza tabela inversa (Inverse S-box) para reverter a substituição aplicada na criptografia.

TB_SUBBYTE E TB_INV_SUBBYTE

TB_SubBytes

```
# == Teste SubBytes ==
# Entrada : 00102030405060708090a0b0c0d0e0f0
# SaÃ-da   : 63cab7040953d051cd60e0e7ba70e18c
# Entrada : 89d810e8855ace682d1843d8cb128fe4
# SaÃ-da   : a761ca9b97be8b45d8ad1a611fc97369
# Entrada : 4915598f55e5d7a0daca94fa1f0a63f7
# SaÃ-da   : 3b59cb73fcd90ee05774222dc067fb68
# Entrada : fa636a2825b339c940668a3157244d17
# SaÃ-da   : 2dfb02343f6d12dd09337ec75b36e3f0
# Entrada : 247240236966b3fa6ed2753288425b6c
# SaÃ-da   : 36400926f9336d2d9fb59d23c42c3950
# Entrada : c81677bc9b7ac93b25027992b0261996
# SaÃ-da   : e847f56514dadde23f77b64fe7f7d490
# Entrada : c62fe109f75eedc3cc79395d84f9cf5d
# SaÃ-da   : b415f8016858552e4bb6124c5f998a4c
# Entrada : d1876c0f79c4300ab45594add66ff41f
# SaÃ-da   : 3e175076b61c04678dfc2295f6a8bfc0
# Entrada : fde3bad205e5d0d73547964ef1fe37f1
# SaÃ-da   : 5411f4b56bd9700e96a0902fa1bb9aa1
# Entrada : bd6e7c3df2b5779e0b61216e8b10b689
# SaÃ-da   : 7a9f102789d5f50b2beffd9f3dca4ea7
# Entrada : 00000000000000000000000000000000
# SaÃ-da   : 63636363636363636363636363636363
# Entrada : ffffffffffffffffffffffffffffffff
# SaÃ-da   : 16161616161616161616161616161616
```

TB_Inv_SubBytes:

```
# == Teste InvSubBytes ==
# Entrada : 7a9f102789d5f50b2beffd9f3dca4ea7
# SaÃ-da   : bd6e7c3df2b5779e0b61216e8b10b689
# Entrada : 5411f4b56bd9700e96a0902fa1bb9aa1
# SaÃ-da   : fde3bad205e5d0d73547964ef1fe37f1
# Entrada : 3e175076b61c04678dfc2295f6a8bfc0
# SaÃ-da   : d1876c0f79c4300ab45594add66ff41f
# Entrada : b415f8016858552e4bb6124c5f998a4c
# SaÃ-da   : c62fe109f75eedc3cc79395d84f9cf5d
# Entrada : e847f56514dadde23f77b64fe7f7d490
# SaÃ-da   : c81677bc9b7ac93b25027992b0261996
# Entrada : 36400926f9336d2d9fb59d23c42c3950
# SaÃ-da   : 247240236966b3fa6ed2753288425b6c
# Entrada : 2dfb02343f6d12dd09337ec75b36e3f0
# SaÃ-da   : fa636a2825b339c940668a3157244d17
# Entrada : 3b59cb73fcd90ee05774222dc067fb68
# SaÃ-da   : 4915598f55e5d7a0daca94fa1f0a63f7
# Entrada : a761ca9b97be8b45d8ad1a611fc97369
# SaÃ-da   : 89d810e8855ace682d1843d8cb128fe4
# Entrada : 63cab7040953d051cd60e0e7ba70e18c
# SaÃ-da   : 00102030405060708090a0b0c0d0e0f0
# Entrada : 00000000000000000000000000000000
# SaÃ-da   : 52525252525252525252525252525252
# Entrada : ffffffffffffffffffffffffffffffff
# SaÃ-da   : 7d7d7d7d7d7d7d7d7d7d7d7d7d7d7d7d
```

EXPANSIONKEY

Módulo responsável por gerar as 11 subchaves (44 palavras de 32 bits) a partir da chave inicial de 128 bits.

- Lógica combinacional (sem clock).
- Usa array local w[0:43] para armazenar as palavras.
- Aplica RotWord, SubWord e XOR com Rcon.
- Saída final: vetor de 1408 bits (44 × 32 bits).
- Usado tanto na criptografia quanto na decriptografia (com vetor invertido).

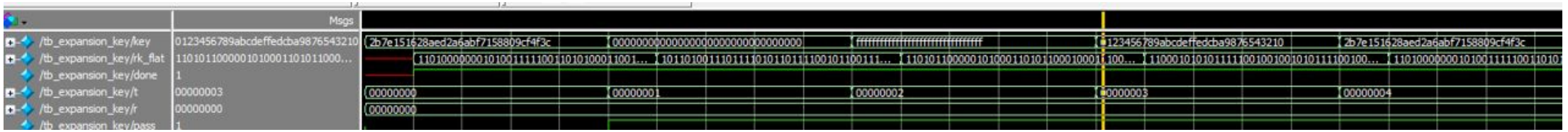
j	$Rcon[j]$	j	$Rcon[j]$
1	[01,00,00,00]	6	[20,00,00,00]
2	[02,00,00,00]	7	[40,00,00,00]
3	[04,00,00,00]	8	[80,00,00,00]
4	[08,00,00,00]	9	[1b,00,00,00]
5	[10,00,00,00]	10	[36,00,00,00]

$$\text{RotWord}([a_0,a_1,a_2,a_3]) = [a_1,a_2,a_3,a_0]$$

$$\text{SubWord}([a_0,\ldots,a_3]) = [\text{SBox}(a_0),\text{SBox}(a_1),\text{SBox}(a_2),\text{SBox}(a_3)]$$

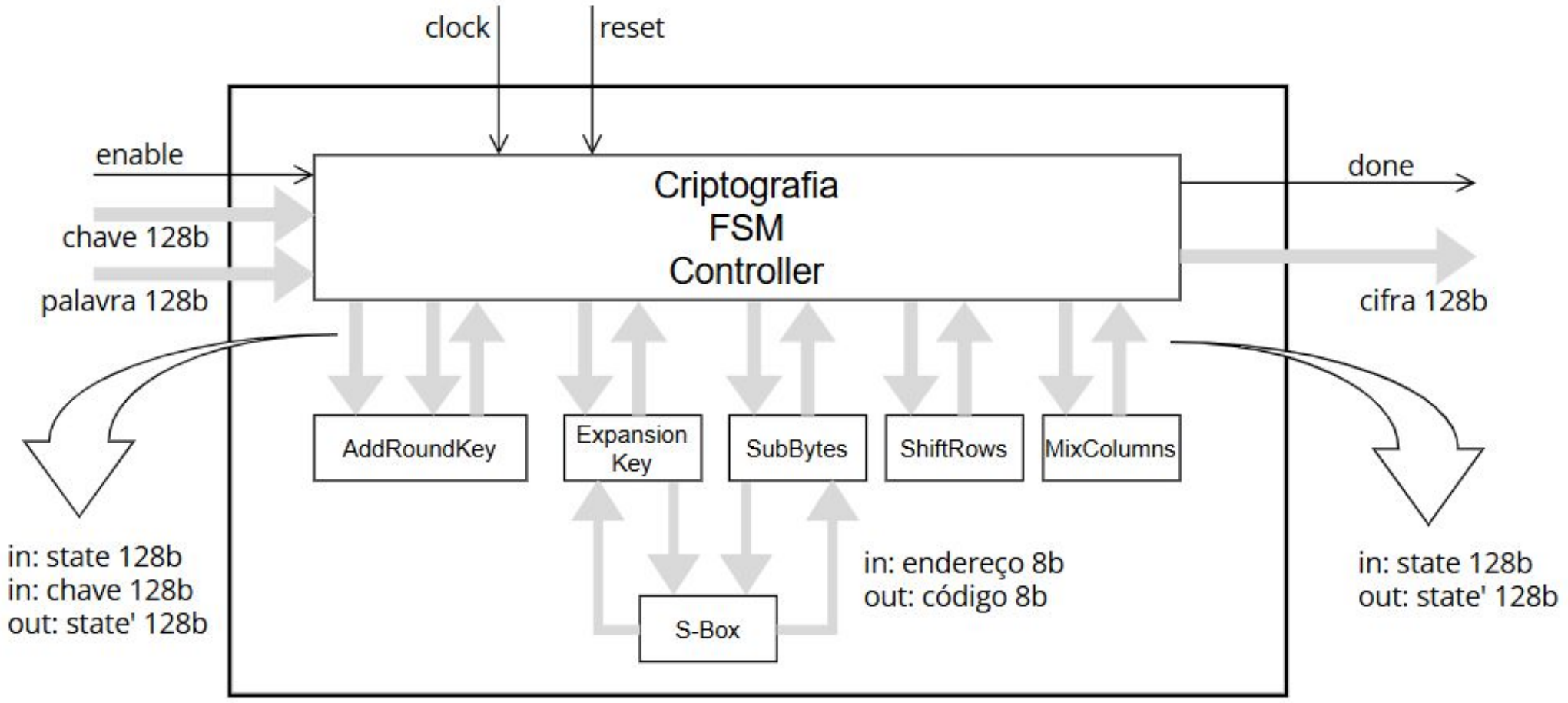
TB_EXPANSIONKEY

Validação feita com valores reais e já testados, o teste número 5 é uma falha intencional

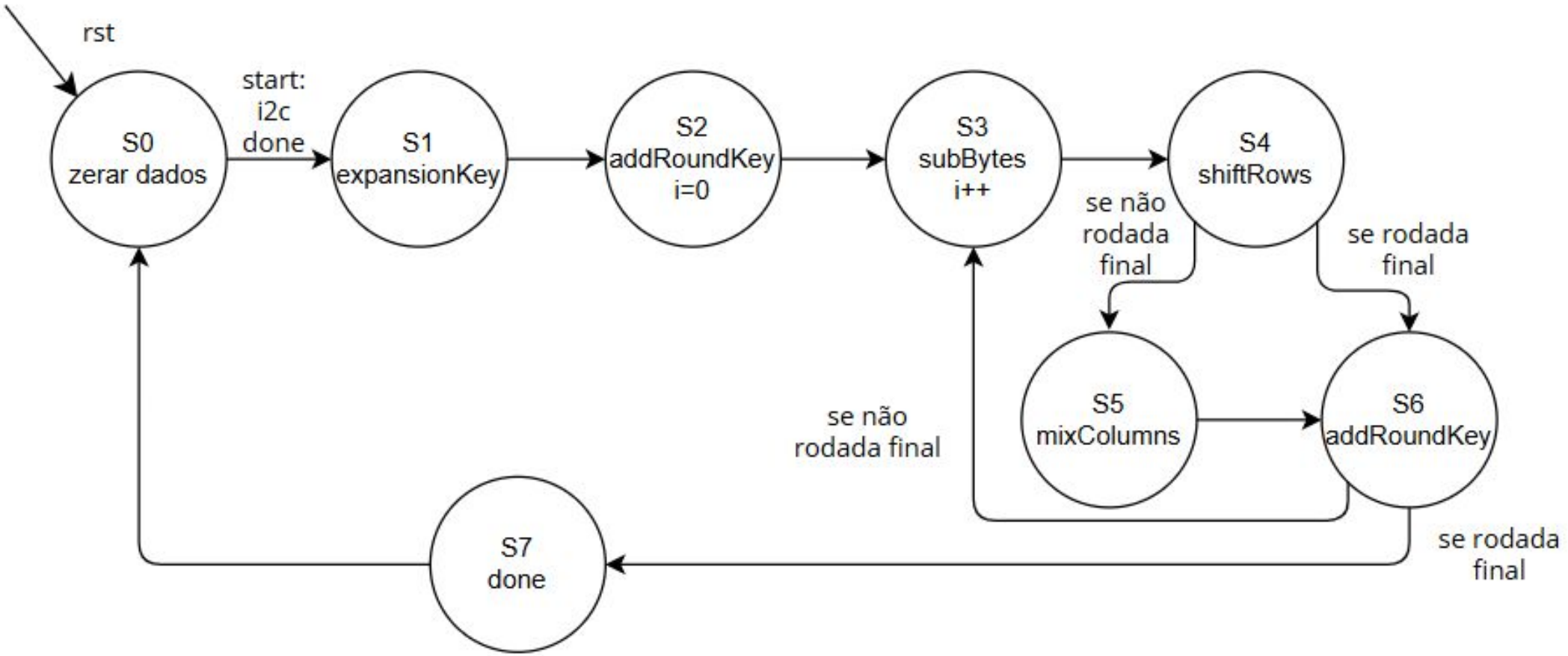



```
# >>> S-Box carregada
#
# ===== BATERIA DE TESTES EXPANSAO AES-128 =====
#
# Caso 0 â chave = 2b7e151628aed2a6abf7158809cf4f3c : PASSOU
#
# Caso 1 â chave = 00000000000000000000000000000000 : PASSOU
#
# Caso 2 â chave = ffffffffffffffffffffffffffffffff : PASSOU
#
# Caso 3 â chave = 0123456789abcdeffedcba9876543210 : PASSOU
#
# Caso 4 â chave = 2b7e151628aed2a6abf7158809cf4f3c : FALHOU
#   Diferenca round 5:
#     obt=d4d1c6f87c839d87caf2b8bc11f915bc
#     exp=0a7c7817a22e2368145f0653cf54ab53
```

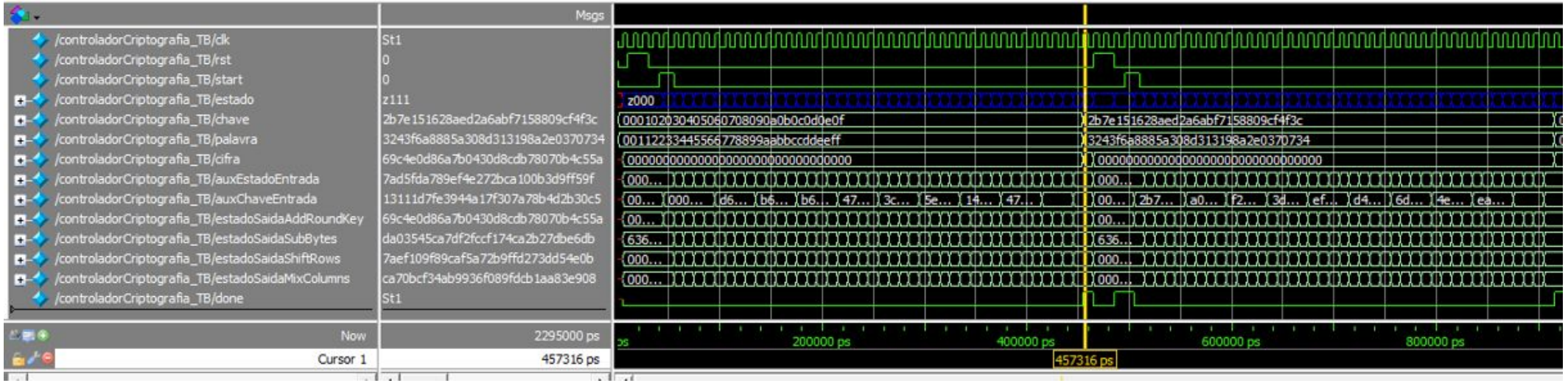
Controller Encriptador



Controller Encriptador



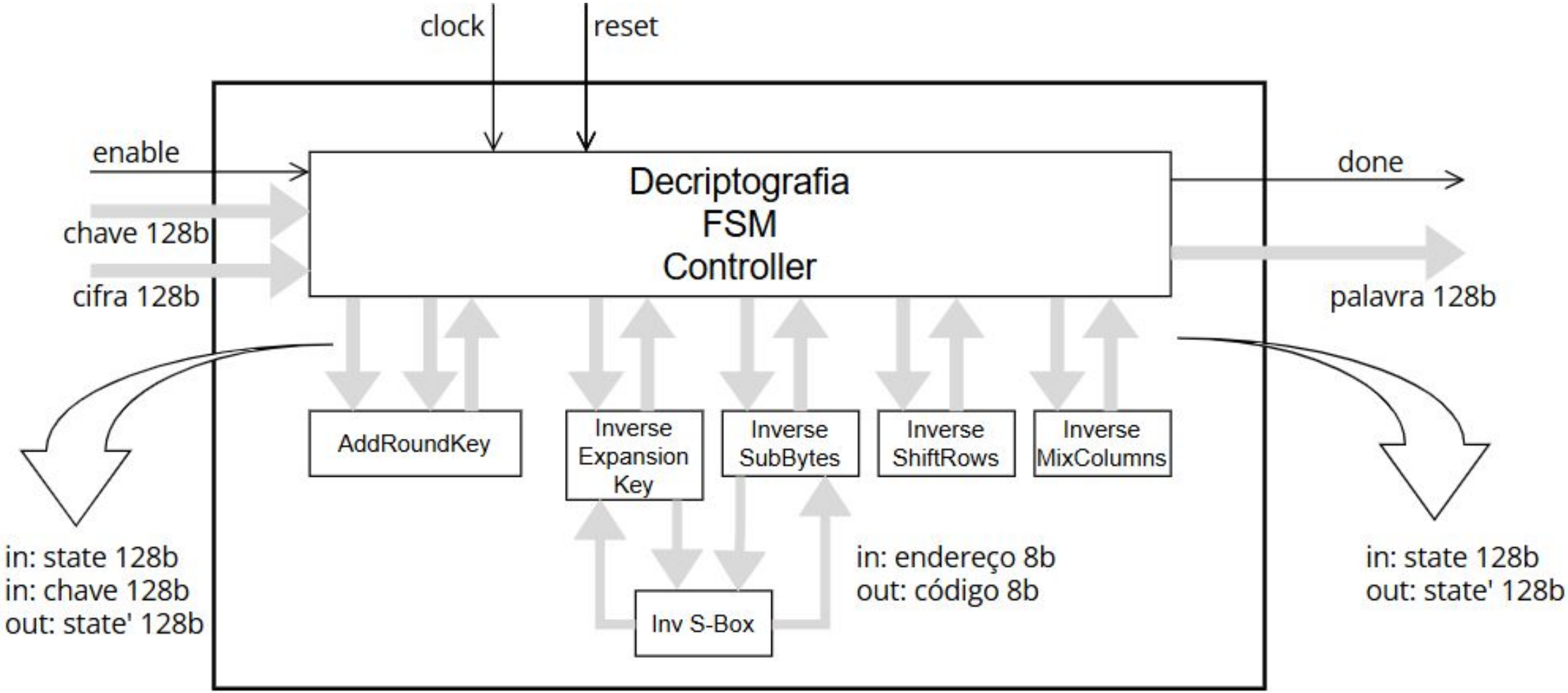
TB_Controller Encriptador



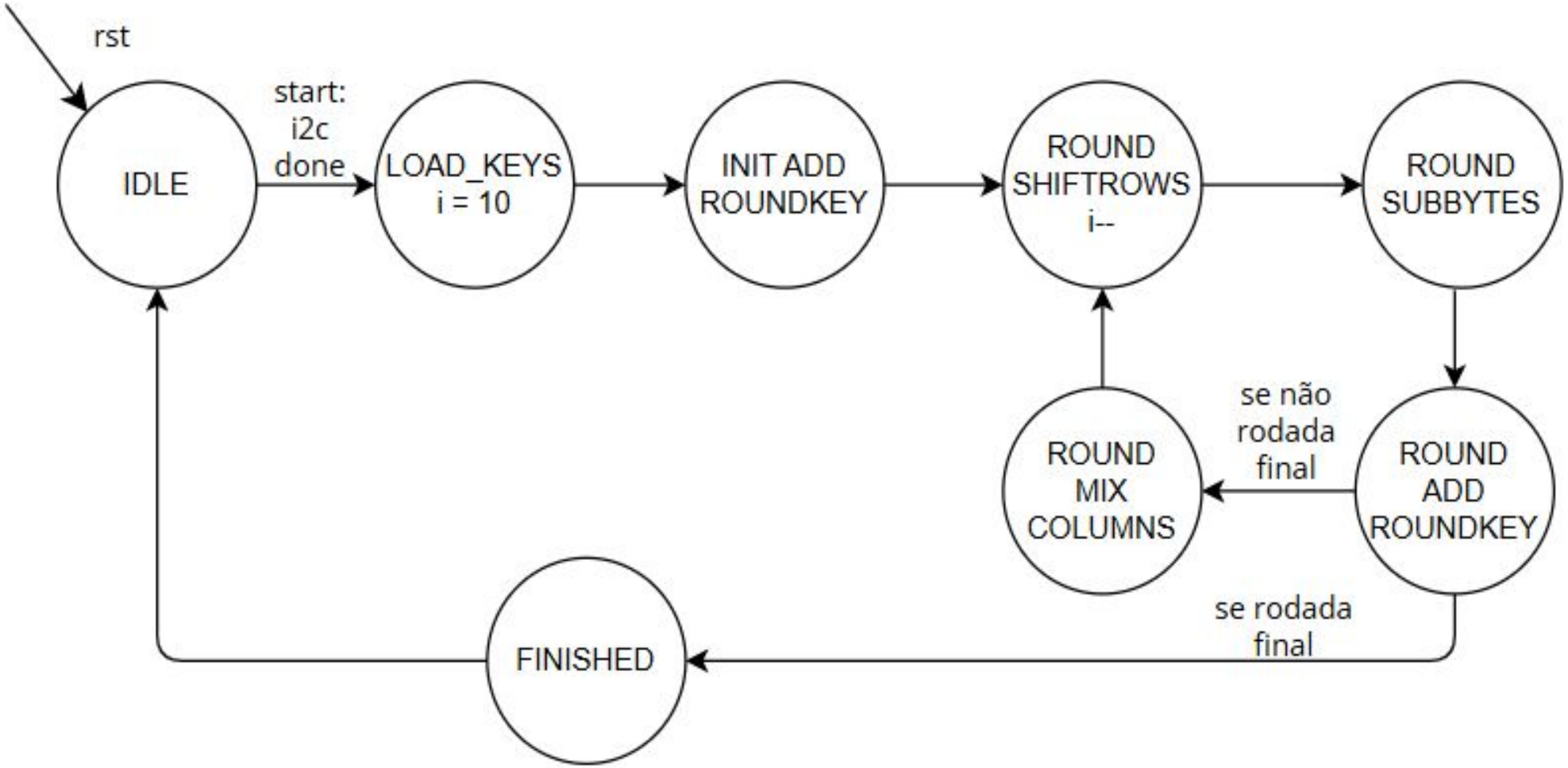

```
# Cifra esperada: 69c4e0d86a7b0430d8cdb78070b4c55a
# Cifra obtida  : 69c4e0d86a7b0430d8cdb78070b4c55a
# â Teste 0 PASSOU
# ------------------------------------------------
#
#
# ------------------------------------------------
# ð Teste 1
# Cifra esperada: 3925841d02dc09fbdc118597196a0b32
# Cifra obtida  : 3925841d02dc09fbdc118597196a0b32
# â Teste 1 PASSOU
# ------------------------------------------------
#
```

```
# ------------------------------------------------
# ð Teste 2
# Cifra esperada: 66e94bd4ef8a2c3b884cfa59ca342b2e
# Cifra obtida  : 66e94bd4ef8a2c3b884cfa59ca342b2e
# â Teste 2 PASSOU
# ------------------------------------------------
#
#
# ------------------------------------------------
# ð Teste 3
# Cifra esperada: 3f5b8cc9ea855a0afa7347d23e8d664e
# Cifra obtida  : 3f5b8cc9ea855a0afa7347d23e8d664e
# â Teste 3 PASSOU
# ------------------------------------------------
#
#
# ------------------------------------------------
# ð Teste 4
# Cifra esperada: 2be52b98821c28a467897944fa4aclbc
# Cifra obtida  : 2be52b98821c28a467897944fa4aclbc
# â Teste 4 PASSOU
# ------------------------------------------------
```

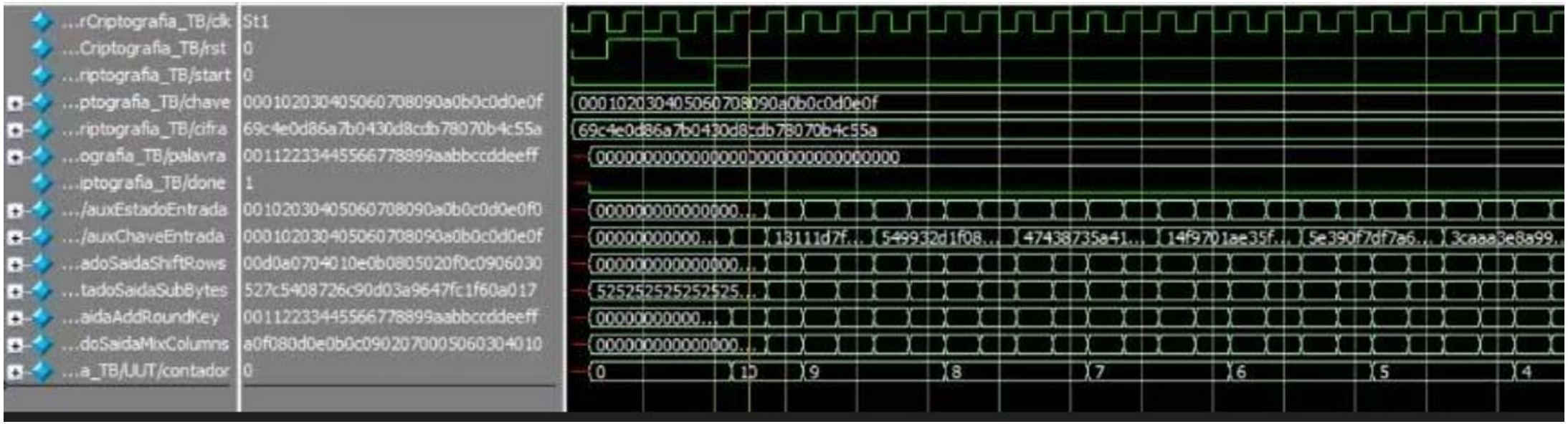
Controller Decriptador



Controller Decriptador



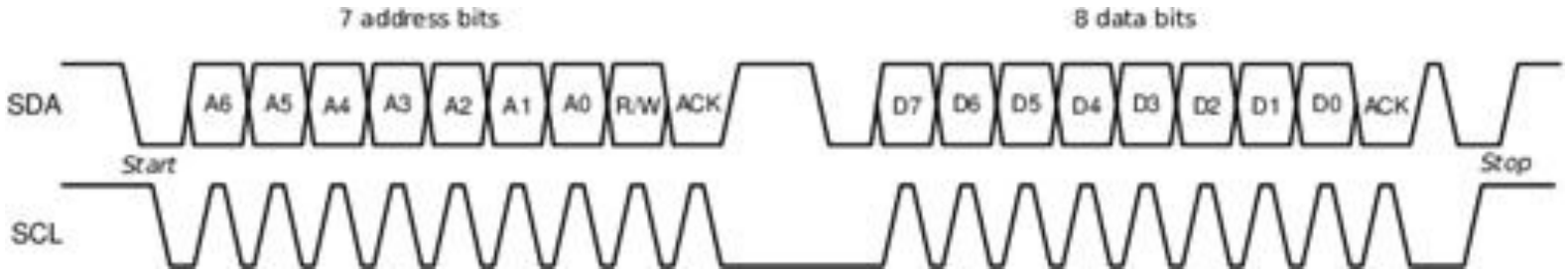
TB_Controller Decriptador



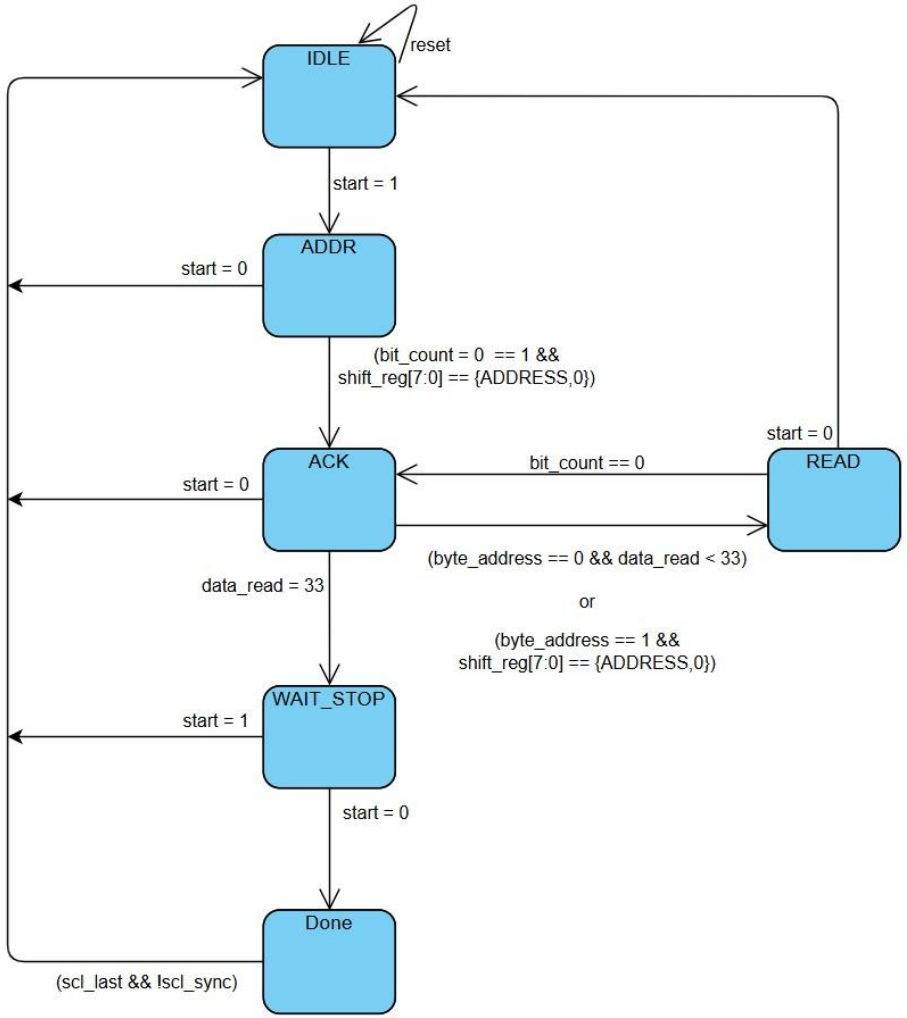
I2C SLAVE

O módulo I2C slave recebe a chave AES do seu mestre - a testbench do I2C, junto com a palavra e o código da operação via o protocolo I2C, operando com FSM e registradores de deslocamento.

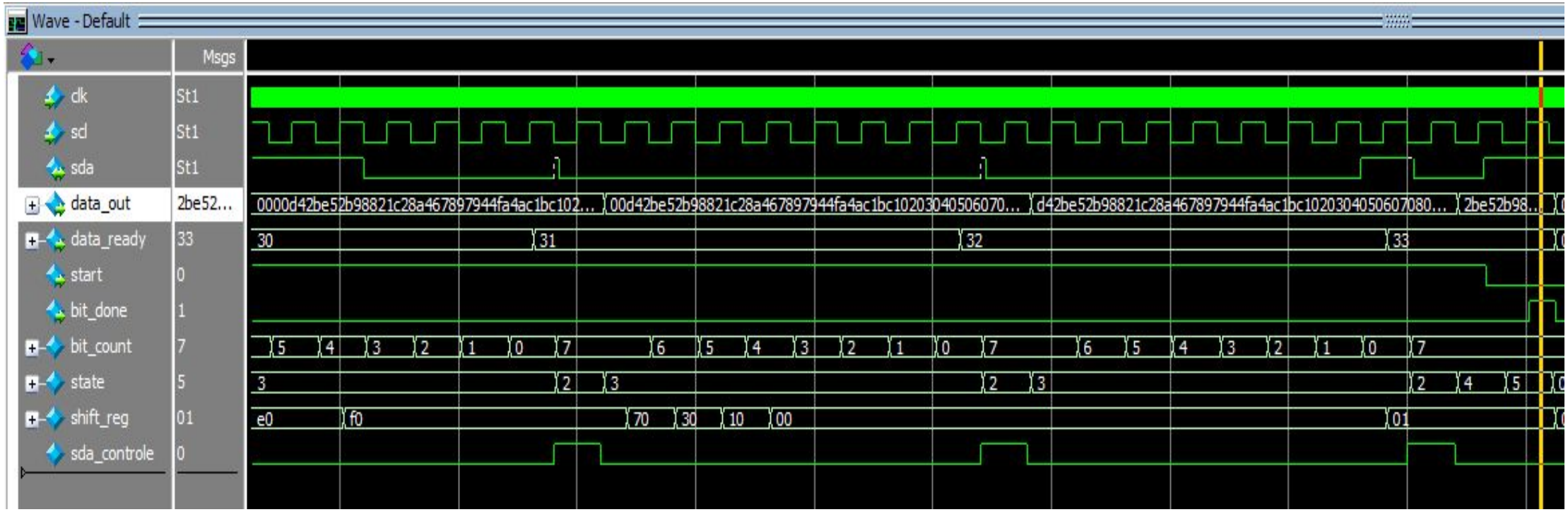
- A comunicação é feita pelas linhas SDA e SCL.
- FSM detecta condições de START/STOP e controla o fluxo.
- Quando o mestre envia um sinal de START, seguido de um endereço, o slave verifica se o endereço é dele.
- Se for, responde com ACK (acknowledge) e inicia o recebimento dos dados.
- Após responder com um último ACK, o mestre finaliza a comunicação com um sinal de STOP.
- Shift registers convertem dados serializados em paralelo.
- Armazena os 264 bits recebidos e notifica o Top Controller.



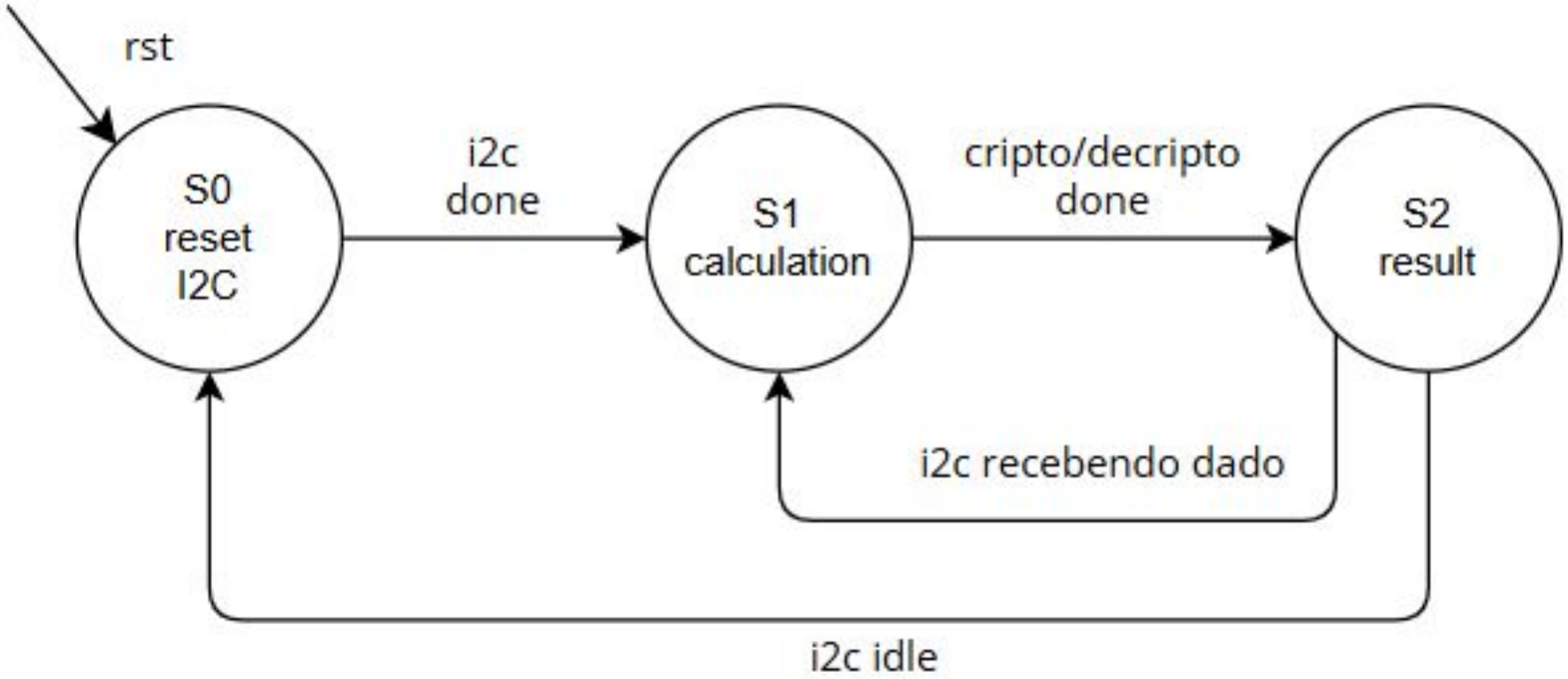
I2C SLAVE


reset
IDLE
start = 1
ADDR
start = 0
(bit_count = 0 == 1 &&
shift_reg[7:0] == {ADDRESS,0})
start = 0
ACK
bit_count == 0
READ
start = 0
(byte_address == 0 && data_read < 33)
or
(byte_address == 1 &&
shift_reg[7:0] == {ADDRESS,0})
data_read = 33
WAIT_STOP
start = 1
start = 0
Done
(scl_last && !scl_sync)

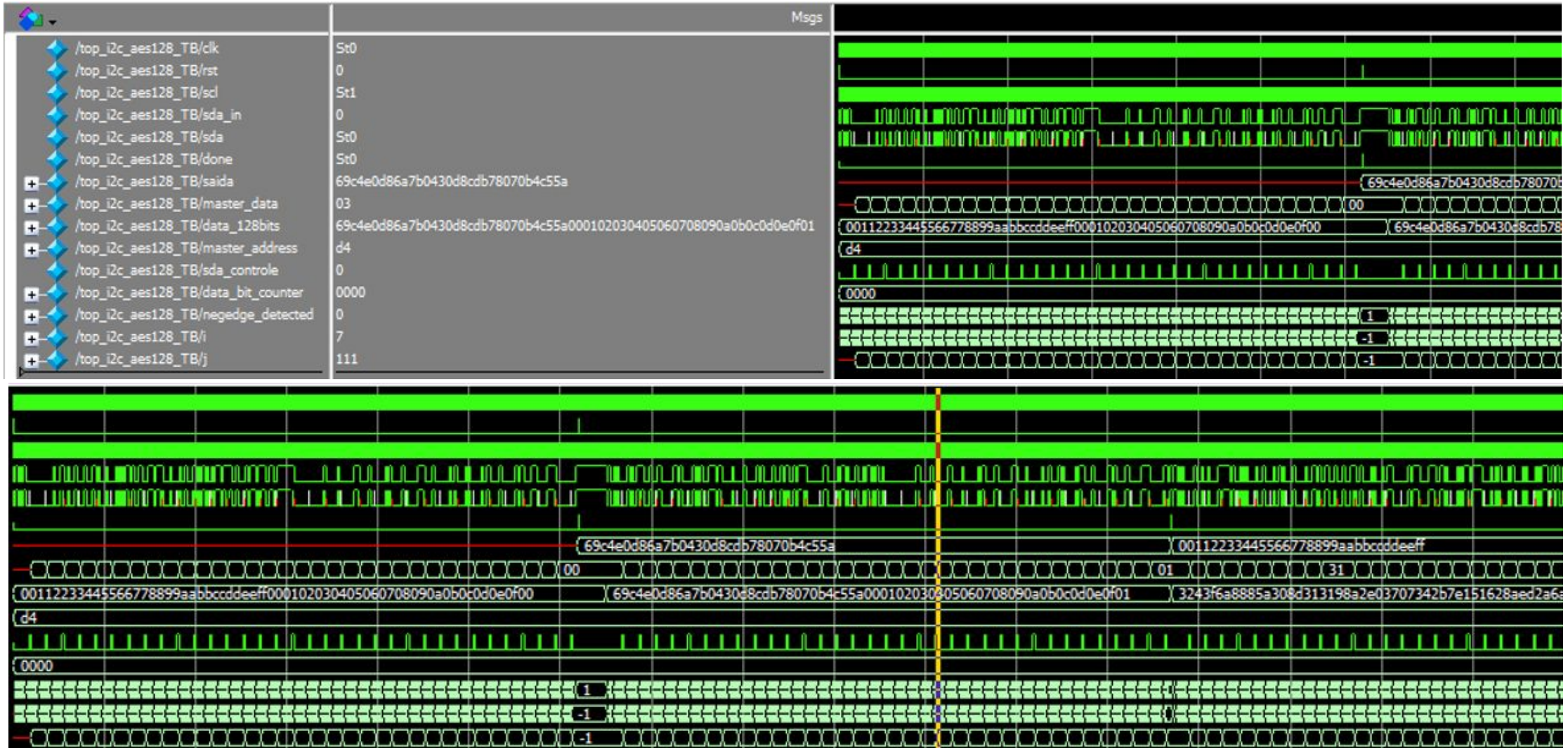
TB_I2C SLAVE


Wave - Default
Msgs
clk
scl
sda
data_out
data_ready
start
bit_done
bit_count
state
shift_reg
sda_controle
St1
St1
St1
2be52...
33
0
1
7
5
3
01
0
0000d42be52b98821c28a467897944fa4ac1bc102...
00d42be52b98821c28a467897944fa4ac1bc10203040506070...
d42be52b98821c28a467897944fa4ac1bc1020304050607080...
2be52b98...
30
31
32
33
e0
f0
70
30
10
00

TOP CONTROLLER



TB_TOP CONTROLLER

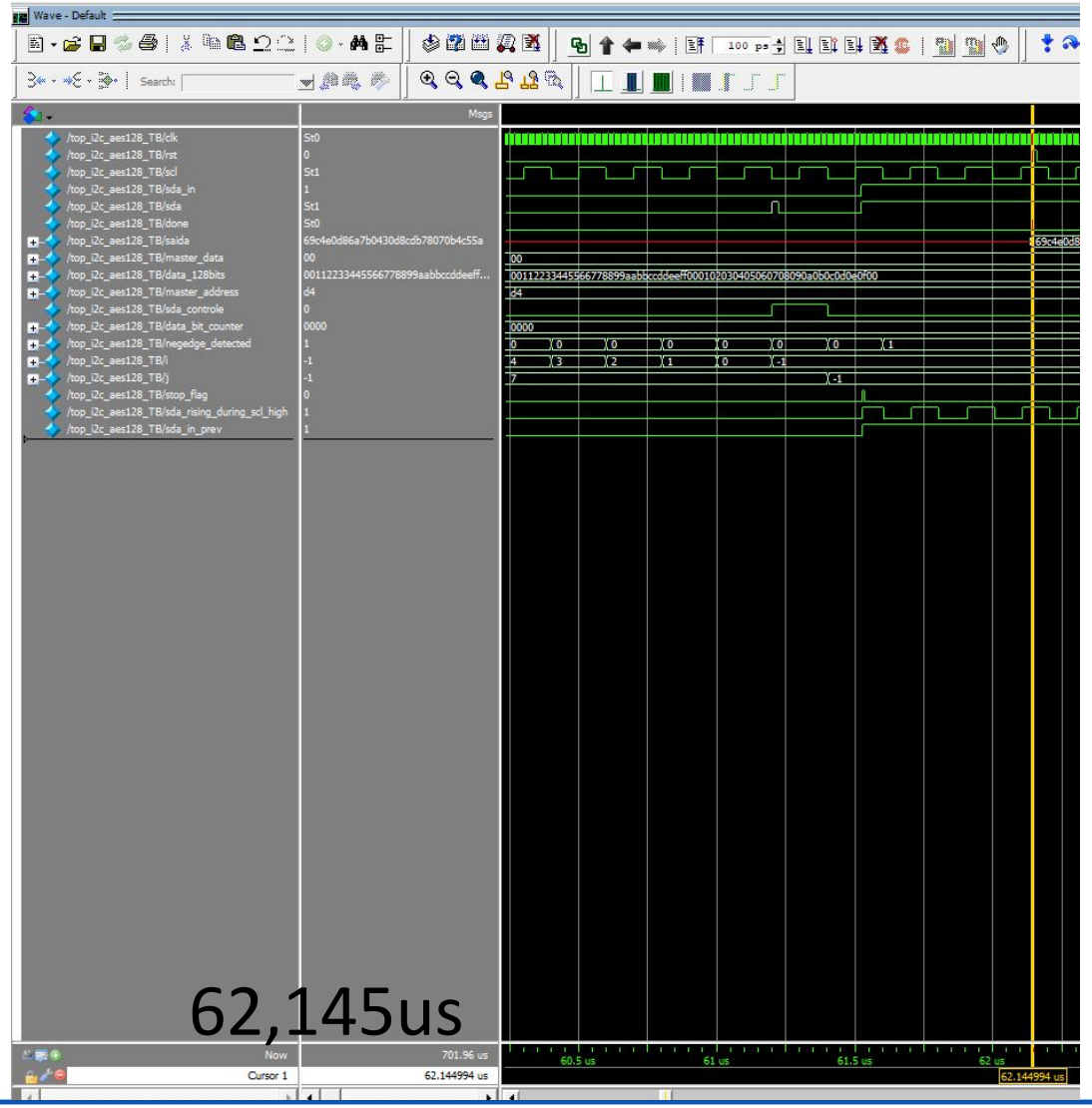


TB_TOP CONTROLLER

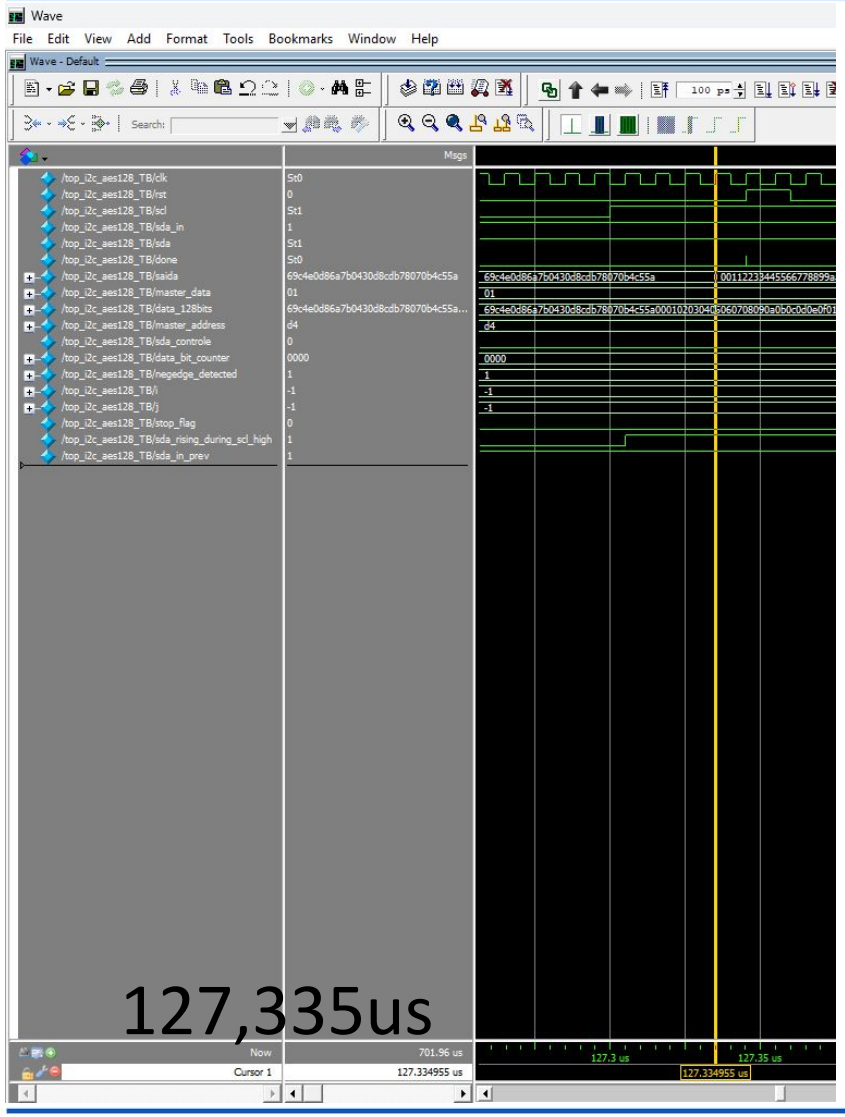
```
VSIM 3> run -all
#
#  Teste 0  Criptografia AES-128
# Entrada  = 00112233445566778899aabbccddeeff
# Chave    = 000102030405060708090a0b0c0d0e0f
# Esperado = 69c4e0d86a7b0430d8cdb78070b4c55a
# Obtido   = 69c4e0d86a7b0430d8cdb78070b4c55a
# Resultado CORRETO!
#
#  Teste 0  Descriptografia AES-128
# Entrada  = 69c4e0d86a7b0430d8cdb78070b4c55a
# Chave    = 000102030405060708090a0b0c0d0e0f
# Esperado = 00112233445566778899aabbccddeeff
# Obtido   = 00112233445566778899aabbccddeeff
# Resultado CORRETO!
#
#  Teste 1  Criptografia AES-128
# Entrada  = 3243f6a8885a308d313198a2e0370734
# Chave    = 2b7e151628aed2a6abf7158809cf4f3c
# Esperado = 3925841d02dc09fbdc118597196a0b32
# Obtido   = 3925841d02dc09fbdc118597196a0b32
# Resultado CORRETO!
#
#  Teste 1  Descriptografia AES-128
# Entrada  = 3925841d02dc09fbdc118597196a0b32
# Chave    = 2b7e151628aed2a6abf7158809cf4f3c
# Esperado = 3243f6a8885a308d313198a2e0370734
# Obtido   = 3243f6a8885a308d313198a2e0370734
# Resultado CORRETO!
#
#  Teste 2  Criptografia AES-128
# Entrada  = 00000000000000000000000000000000
# Chave    = 00000000000000000000000000000000
# Esperado = 66e94bd4ef8a2c3b884cfa59ca342b2e
# Obtido   = 66e94bd4ef8a2c3b884cfa59ca342b2e
# Resultado CORRETO!
#
#  Teste 2  Descriptografia AES-128
# Entrada  = 66e94bd4ef8a2c3b884cfa59ca342b2e
# Chave    = 00000000000000000000000000000000
# Esperado = 00000000000000000000000000000000
# Obtido   = 00000000000000000000000000000000
# Resultado CORRETO!
#
```

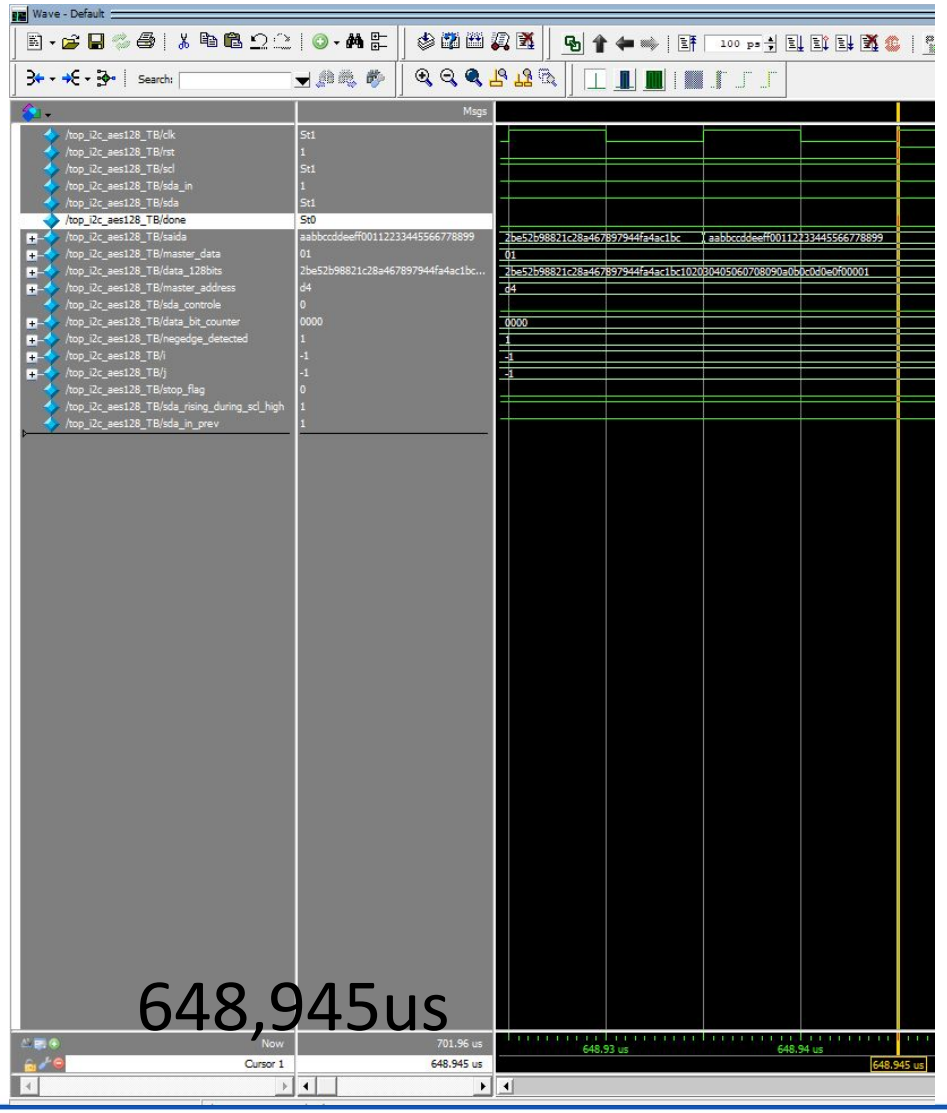
TB_TOP CONTROLLER

```
#  Teste 3  Criptografia AES-128
# Entrada  = ffffffffffffffffffffffffffffffff
# Chave    = 00000000000000000000000000000000
# Esperado = 3f5b8cc9ea855a0afa7347d23e8d664e
# Obtido   = 3f5b8cc9ea855a0afa7347d23e8d664e
# Resultado CORRETO!
#
#  Teste 3  Descriptografia AES-128
# Entrada  = 3f5b8cc9ea855a0afa7347d23e8d664e
# Chave    = 00000000000000000000000000000000
# Esperado = ffffffffffffffffffffffffffffffff
# Obtido   = ffffffffffffffffffffffffffffffff
# Resultado CORRETO!
#
#  Teste 4  Criptografia AES-128
# Entrada  = aabbccddeeff00112233445566778899
# Chave    = 102030405060708090a0b0c0d0e0f000
# Esperado = 2be52b98821c28a467897944fa4ac1bc
# Obtido   = 2be52b98821c28a467897944fa4ac1bc
# Resultado CORRETO!
#
#  Teste 4  Descriptografia AES-128
# Entrada  = 2be52b98821c28a467897944fa4ac1bc
# Chave    = 102030405060708090a0b0c0d0e0f000
# Esperado = aabbccddeeff00112233445566778899
# Obtido   = aabbccddeeff00112233445566778899
# Resultado CORRETO!
```



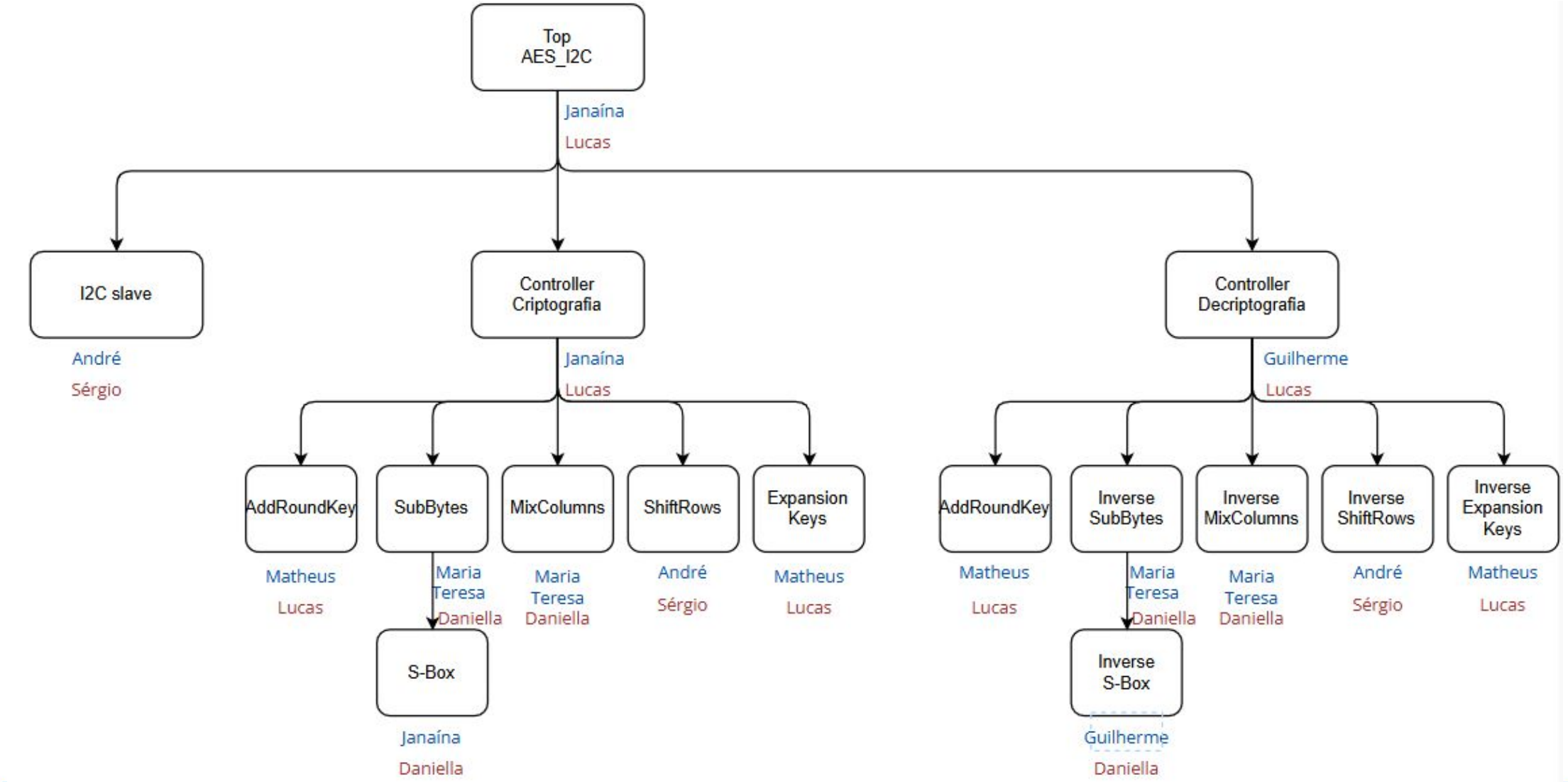
TB_TOP CONTROLLER





CONCLUSÃO

CONCLUSÃO


Top
AES_I2C
Janaína
Lucas
I2C slave
André
Sérgio
Controller
Criptografia
Janaína
Lucas
Controller
Decriptografia
Guilherme
Lucas
AddRoundKey
Matheus
Lucas
SubBytes
Maria
Teresa
Daniella
MixColumns
Maria
Teresa
Daniella
ShiftRows
André
Sérgio
Expansion
Keys
Matheus
Lucas
S-Box
Janaína
Daniella
AddRoundKey
Matheus
Lucas
Inverse
SubBytes
Maria
Teresa
Daniella
Inverse
MixColumns
Maria
Teresa
Daniella
Inverse
ShiftRows
André
Sérgio
Inverse
Expansion
Keys
Matheus
Lucas
Inverse
S-Box
Guilherme
Daniella

CONCLUSÃO

Pontos Fracos

- Dificuldade inicial de entendimento do conteúdo (tanto da parte matemática quanto da parte conceitual da criptografia)
- Dificuldade em dividir o projeto

Pontos Fortes

- Tempo inicial dedicado ao estudo do tema
- Divisão em pequenos módulos
- Mesma equipe de designer e tester para módulos de operações inversas
- Projeto bem estruturado e com grande quantidade de testes
- Comprometimento da equipe com as entregas e prazos
- Apoio entre a equipe para resolução de problemas ao longo do desenvolvimento