

Guia 4

I2C: Interface com EEPROM e encriptação de dados

Trabalho realizado por:

- ➔ João Peixoto nº 91960
- ➔ João Rocha nº 91936

1. Breve Introdução

Neste trabalho pretende-se ler e escrever mensagens de texto encriptadas num periférico I2C (memória E2PROM). As mensagens antes de serem armazenadas na memória EEPROM devem ser previamente encriptadas com uma chave de encriptação de 4 bytes e o seu conteúdo só estará legível depois da sua desencriptação, impedindo assim que os dados transmitidos possam ser interpretados por terceiros.

2. Desafio

Fomos desafiados a implementar um programa para o microprocessador C8051F388 que permita ler e escrever dados encriptados numa memória EEPROM da família 24CXX (24C01/2/4/8/16), recebidos pela porta série.

Apesar de existir um periférico two-wire interface TWI (que implementa o protocolo I2C) no microprocessador C8051F388, este não é suportado pelo simulador do Keil, pelo que o programa a desenvolver deve fazer uso dos pinos de GPIO para implementar protocolo através de software, usando uma técnica conhecida como bit banging.

Desta forma é possível a implementação e debug do protocolo através do ambiente de simulação do Keil onde, e com o auxílio da ferramenta “logic analyzer”, torna-se mais fácil a interpretação e visualização das tramas enviadas para a memória através dos pinos GPIO selecionados, bem como a manipulação e simulação dos dados recebidos.

3. Configuração de Periféricos

```
5 // *****
6 // ****                                OSCILATOR INIT                                ****
7 // *****
8
9 void Oscillator_init(void)
10 {
11     FLSCL      = 0x90;
12     CLKSEL     = 0x03;    // SYSCLK a 48MHz
13 }
14
15 // *****
16 // ****                                TIMER INIT                                ****
17 // *****
18
19 void Timer_init(void)
20 {
21     TMOD       = 0x02;    // Timer 0 8bit auto-reload
22     CKCON      = 0x02;    // Prescaled Clock Input -> SYSCLK / 48
23
24     TH0 = {-25};
25     TL0 = {-25};    // T overflow = 25uS -> f = 40KHz
26 }
27
28 // *****
29 // ****                                UART INIT                                ****
30 // *****
31
32 void UART_Init()
33 {
34     SBRL1 = 0x30;
35     SBRLH1 = 0xff;    // BaudRate a 115200 bps
36     SCON1 = 0x10;    // Enable UART1 reception -> REN1
37     SBCON1 = 0x43;
38     EIE2 |= 0x02;    // Enable UART1 Interrupt
39 }
40
41 // *****
42 // ****                                PCA INIT                                ****
43 // *****
44
45 void PCA_init(void)
46 {
47     PCA0MD = 0;    // Disable Watch-Dog Timer
48 }
49
50 // *****
51 // ****                                PORT_IO INIT                                ****
52 // *****
53
54 void Port_IO_init(void)
55 {
56     POSKIP     = 0x0F;    // Tx0 e Rx0 a P0_4 e P0_5
57     XBR1        = 0x40;    // CrossBar Enable
58     XBR2        = 0x01;    // UART1 Enable on CrossBar
59 }
60
61 // *****
62 // ****                                DEVICE INIT                                ****
63 // *****
64
65 void Device_init(void)
66 {
67     Oscillator_init();    // Call Oscillator Init
68     UART_Init();    // Call UART Init
69     PCA_init();    // Call PCA Init
70     Port_IO_init();    // Call Port_IO Init
71     Timer_init();    // Call Timer Init
72 }
73
74
```

4. Conjunto de Variáveis

Tal como podemos ver abaixo, para a implementação do bitbang, seguimos uma abordagem modelar e fizemos uma State Machine. Para tal, criamos um enumerado, para representar todos os Estados possíveis.

Cada transferência (Escrita ou Leitura) está representada numa estrutura de dados própria (I2C_TRANSFER) e, de forma a abstrair o utilizador e a implementar uma 'stack' para o I2C de 2 níveis, todas as transferências realizadas encontram-se guardadas num buffer, podendo assim controlar quantas transferências foram realizadas até ao momento, buscar uma determinada transferência ou até mesmo eliminar uma do seu histórico.

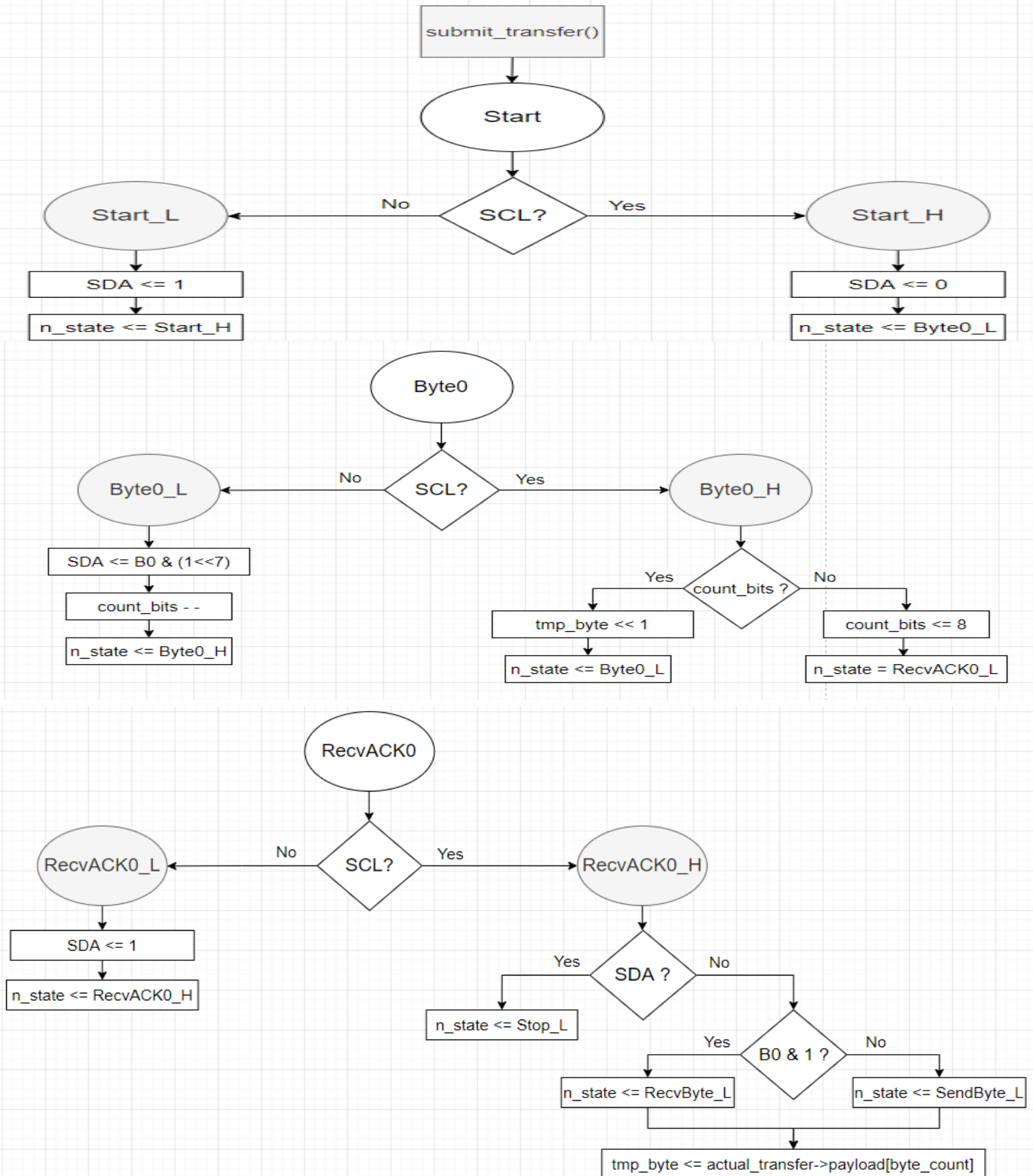
A sua estrutura de denomina-se I2C_TRANSFER_BUFFER.

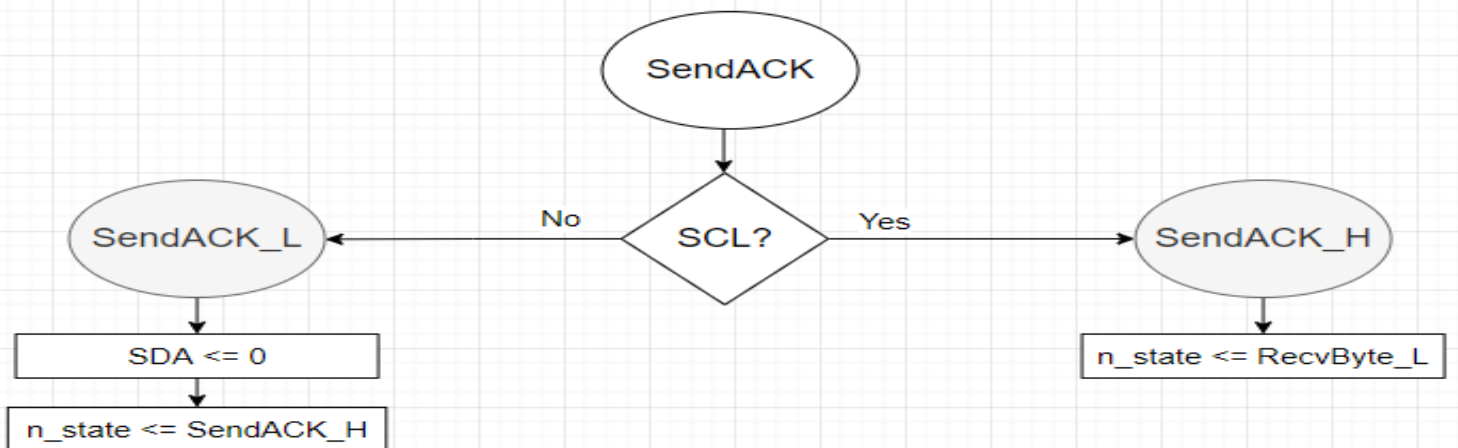
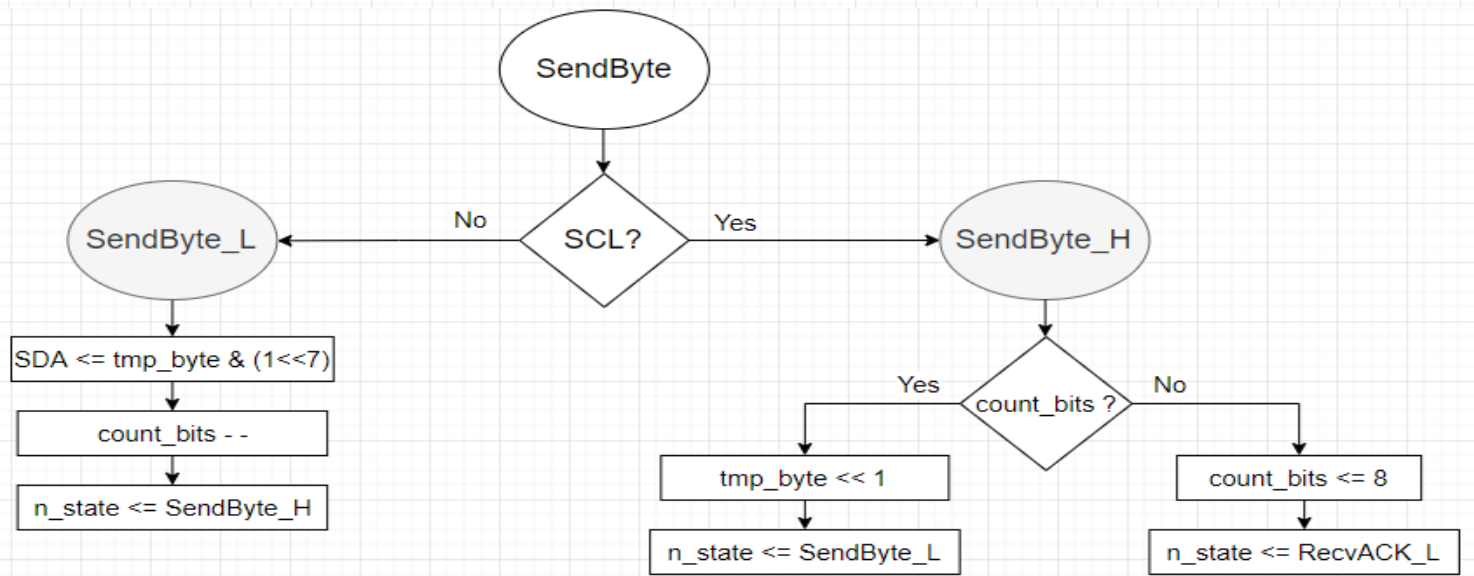
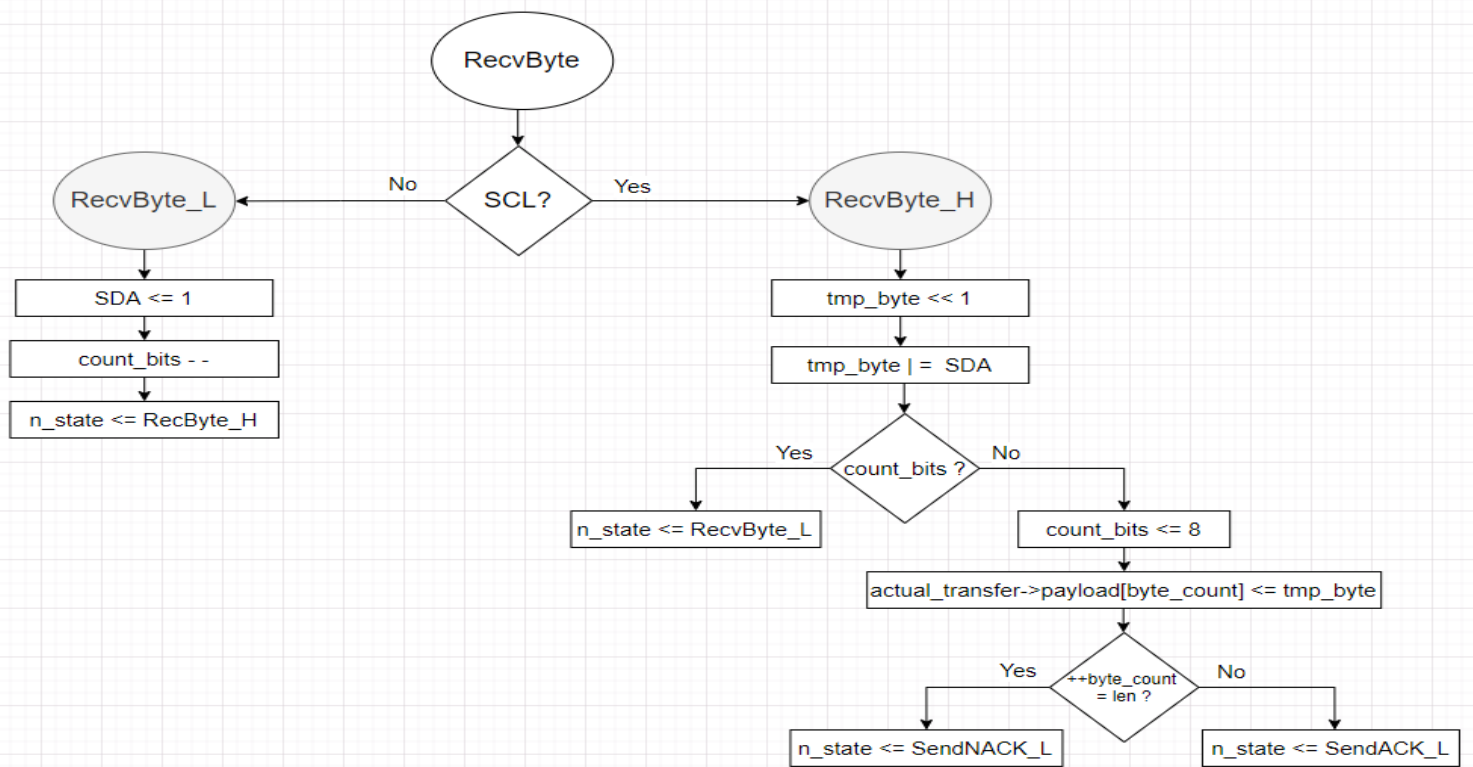
```
5 // *****
6 // *****          STATES ENUMERATION          *****
7 // *****
8
9 typedef enum STATES
10 {
11     Idle = 0,
12     Start_L, Start_H,           // Start State's
13     Byte0_L, Byte0_H,          // Byte0 State's
14     RecvACK0_L, RecvACK0_H,     // Receive ACK0 State's
15     RecvByte_L, RecvByte_H,    // Receive Byte State's
16     SendByte_L, SendByte_H,    // Send Byte State's
17     SendACK_L, SendACK_H,      // Send ACK State's
18     RecvACK_L, RecvACK_H,      // Receive ACK State's
19     SendNACK_L, SendNACK_H,    // Send NACK State's
20     Stop_L, Stop_H            // Stop State's
21 } states_t;
22
23
24 // *****
25 // *****          I2C_TRANSFER STRUCT          *****
26 // *****
27
28 typedef struct I2C_TRANSFER
29 {
30     unsigned char byte0;        // Slave ADDR
31     unsigned char len;          // Number of Bytes to Read/Write
32     unsigned char byte_count;   // For Bytes iteration
33     unsigned char *xdata;       // Buffer: Write - read from here to write | Read - save here
34 } i2c_transfer_t;
35
36
37 // *****
38 // *****          I2C_TRANSFER_BUFFER STRUCT          *****
39 // *****
40
41 typedef struct I2C_TRANSFER_BUFFER
42 {
43     i2c_transfer_t *xdata;      // I2C Transfers Buffer
44     unsigned char start;        // Start Buffer
45     unsigned char end;          // End Buffer
46     unsigned char len;          // Buffer Length
47 } i2c_transfer_buffer_t;
```

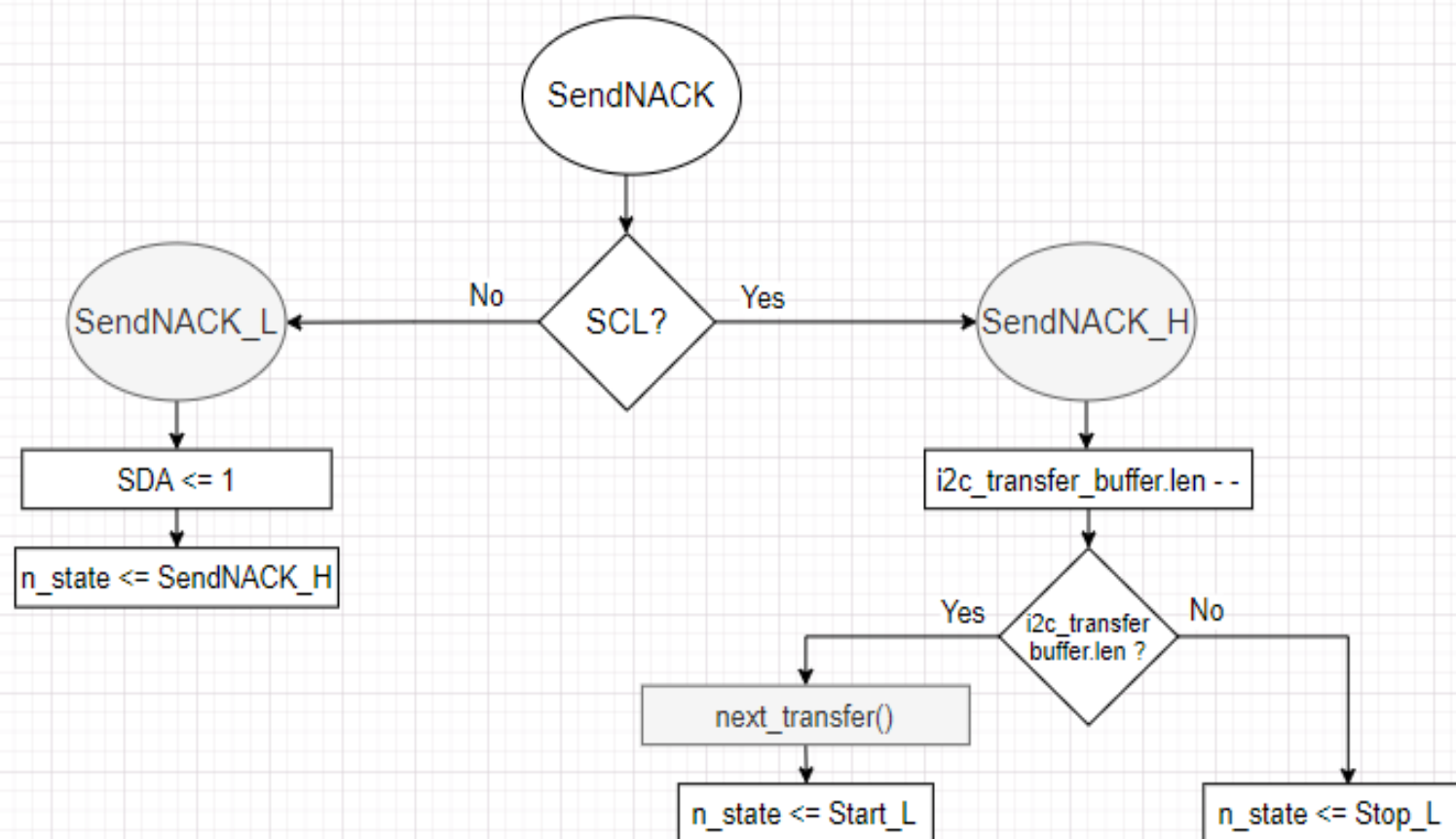
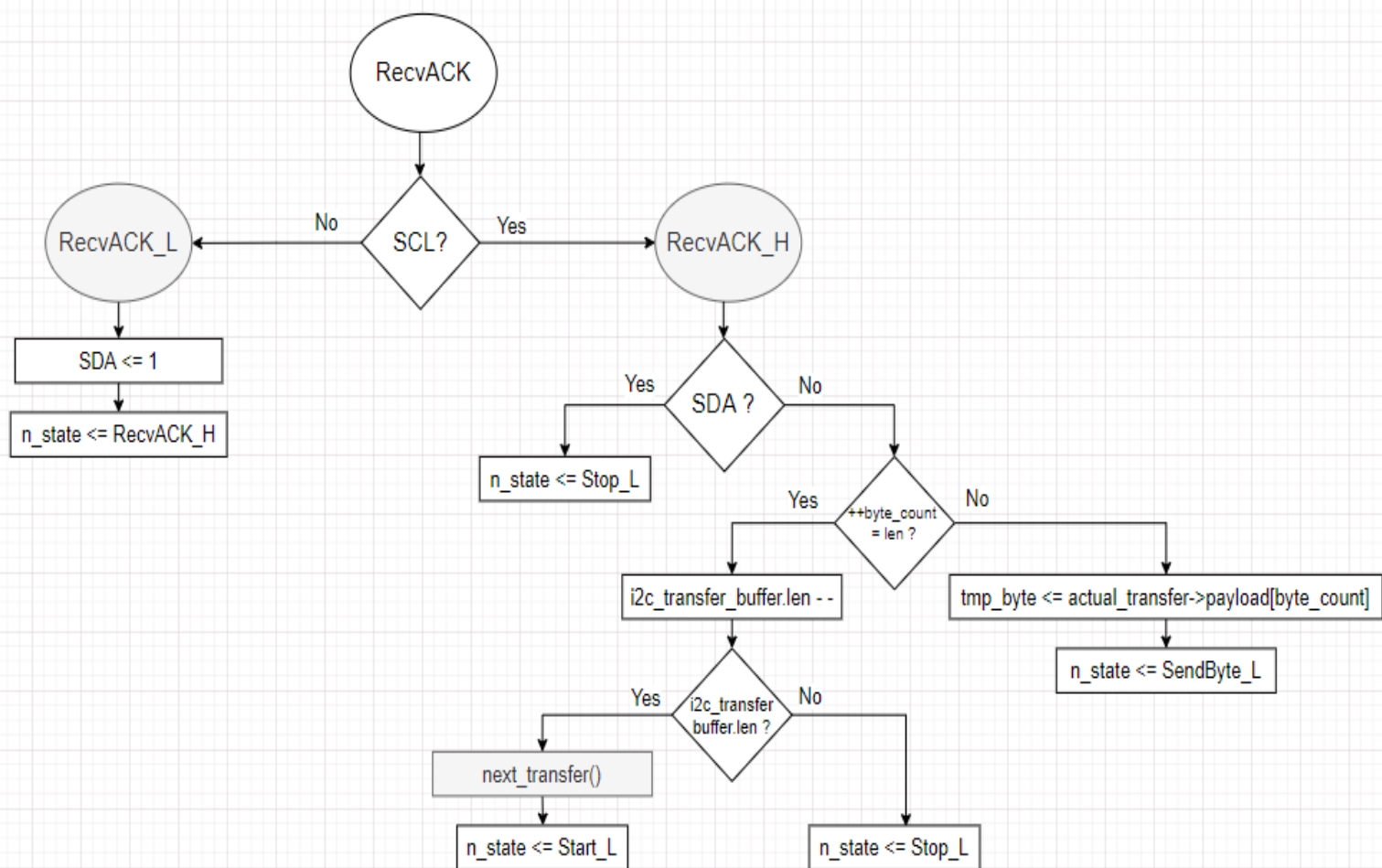
5. Fluxogramas

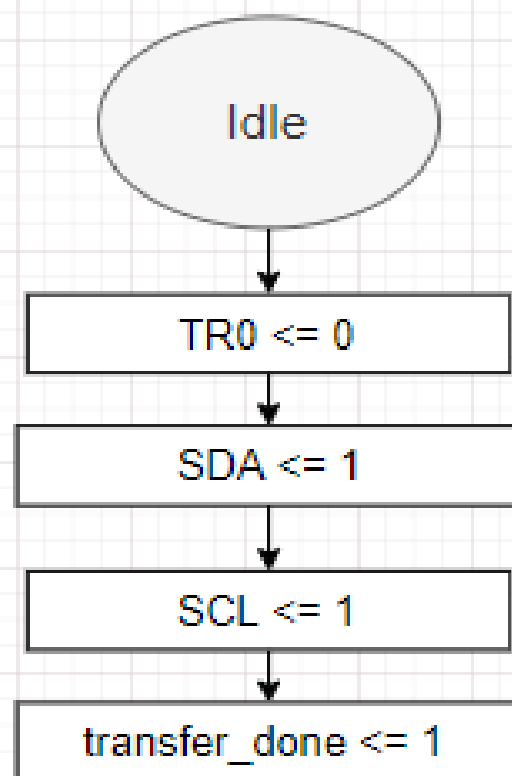
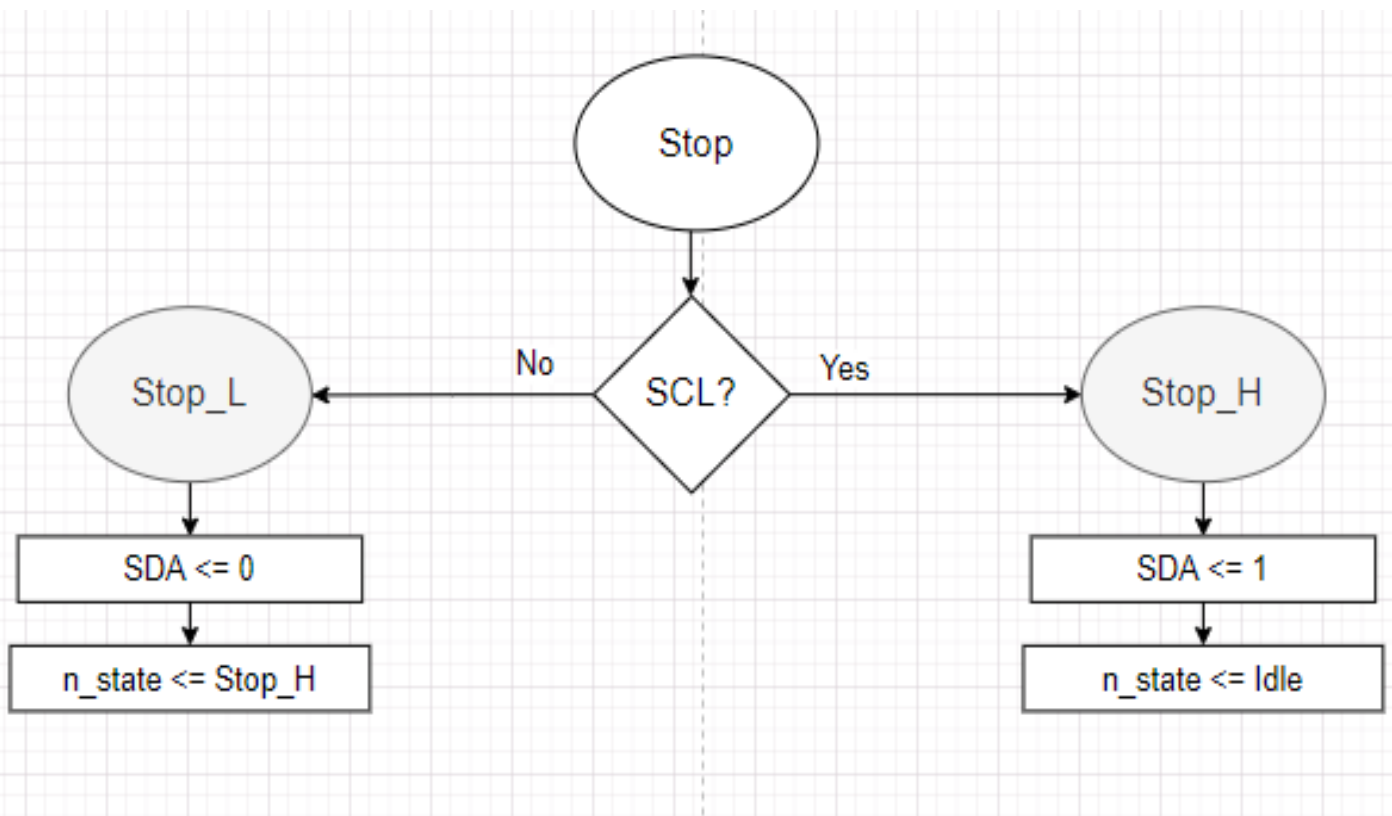
Como dito atrás, implementamos o BitBang recorrendo a uma máquina de Estados.

De forma a explicar e entender melhor como esta Máquina de Estados funciona, em baixo, poderemos observar todos os princípios de funcionamento de cada estado, bem como o fluxo de uma, ou várias, transferência(s).









6. Funções EEPROM

Nesta fase, depois de implementar o 'grau mais baixo', isto é, o bitbang para o I2C, criamos duas funções (Escrita e Leitura), que nos permitisse, no caso da escrita, receber 3 parâmetros (o array / caracteres que queiramos escrever), o número de elementos que queremos escrever e o endereço onde queremos escrever.

No caso da leitura, também recebemos 3 parâmetros, mas em que neste caso é passado um array onde os dados lidos vão ser alojados, o número de elementos que vamos querer ler e o endereço onde queremos ler.

Ambas as funções retornam um valor, em que se este for positivo, corresponde ao número de bytes escritos / lidos com sucesso, emitindo via porta série este número bem como (no caso da leitura), os bytes lidos.

Caso o número seja negativo, é emitida uma mensagem de erro via porta série.

```
1 #ifndef _EEPROM_I2C_
2 #define _EEPROM_I2C_
3
4 // *****
5 // ****                      EEPROM DEFINES                      ****
6 // *****
7
8 #define EEPROM_WR 0xA0
9 #define EEPROM_RD 0xA1
10 #define PAGE_SIZE 16
11 #define PAGE_NUMBER 256
12 #define MAX_LENGTH 64
13 #define CYPHER_LEN 4
14
15 extern char xdata cypher[CYPHER_LEN];
16
17
18 // *****
19 // ****                      EEPROM WRITE FUNCTION                      ****
20 // *****
21 int eeprom_write(char xdata* wdata, unsigned char len, unsigned char addr);
22
23
24 // *****
25 // ****                      EEPROM READ FUNCTION                      ****
26 // *****
27 int eeprom_read(char xdata* rdata, unsigned char len, unsigned char addr);
28
29
30 #endif
```


7. Encriptação de Dados


Como dito no enunciado, a mensagem em texto limpo deverá ser recebida através da interface série, previamente desenvolvido nos guias anteriores, e uma vez armazenada na memória interna do microcontrolador, deve ser encriptada aplicando a chave de encriptação ao array de memória correspondente, antes de ser posteriormente enviada via I2C para armazenamento na memória EEPROM.

Após uma sequência de leitura de uma mensagem armazenada na memória EEPROM, é necessário voltar a aplicar a chave de encriptação de forma a ser possível obter-se novamente a mensagem em texto limpo.

A chave de encriptação é normalmente negociada, de forma segura, numa etapa de handshake da comunicação, mas por questões de simplicidade a respetiva chave deve ser recebida via porta série e ficar armazenada na memória interna do microcontrolador, e com o tamanho predefinido de 4 bytes.

Exemplo de execução e codificação:

```
6 char xdata cypher[CYPHER_LEN] = {'A', 'A', 'A', 'A'};           // Default Cypher Key
7
8 // *****
9 // ****                  CYPHER ALGORITHM                      ****
10 // *****
11
12
13 void encrypt_msg(char xdata* buf, unsigned char buf_len)
14 {
15     unsigned char i;
16     for (i = 0; i < buf_len; i++)
17         buf[i] ^= cypher[i % CYPHER_LEN];                       // Fill and encrypt the buffer with the cypher key
18 }
19
```

 Termite 3.1 (by CompuPhase)


COM4 115200 bps, 8N1, no handshake

```
3
Insert the new Cypher Key:
1 Byte: 1
2 Byte: 2
3 Byte: 3
4 Byte: 4
Cypher key has been updated!
```

8. Fase de Experimentação / Testes


Na fase final, e como objetivo final, a comunicação com uma memória (EEPROM) através do protocolo série I2C, decidimos testar com uma memória (24LC08B), realizando, com sucesso, todas as escritas e leituras realizadas.

Abaixo mostramos algumas imagens dos testes realizados:

 Termite 3.1 (by CompuPhase)

COM4 115200 bps, 8N1, no handshake

```
1
ADDR to write: 0
Insert the number of bytes you are going to write: 2
1ª byte: 8
2ª byte: 9
2 bytes written successfully!
2
ADDR to read: 0
Insert the number of bytes you are going to read: 2
8
9
```

 Termite 3.1 (by CompuPhase)

COM4 115200 bps, 8N1, no handshake

```
3
Insert the new Cypher Key:
1 Byte: 1
2 Byte: 2
3 Byte: 3
4 Byte: 4
Cypher key has been updated!
```

João Peixoto nº 91960

João Rocha nº 91936

Termite 3.1 (by CompuPhase)

COM4 115200 bps, 8N1, no handshake

ADDR to write: 14

Insert the number of bytes you are going to write: 12

1ª byte: 1

2ª byte: 2

3ª byte: 3

4ª byte: 4

5ª byte: 5

6ª byte: 6

7ª byte: 7

8ª byte: 8

9ª byte: 9

10ª byte: 10

11ª byte: 11

12ª byte: 12

12 bytes written successfully!

2

ADDR to read: 14

Insert the number of bytes you are going to read: 12

1

2

3

4

5

6

7

8

9

10

11

12

