Rafael Lemos Pedro

Comunicação serie pic18f46k40 - esp32c3

July 26, 2022

1 Dados parameterizáveis

Os dados parametrizáveis relevantes no *pic18f46k40* estão instanciados na *main.h* e organizados dentro de duas estruturas. A primeira estrutura contém parâmetros registados na memória volátil e é permitida a sua escrita e leitura. Os valores desta estrutura podem ser carregados para a memória não volátil após a sua edição.

```
typedef struct
{
    //page0
    unsigned char decelarationOpen;          // Duration of the
       deceleration phase on the opening process [0..15sec].
    unsigned char decelarationClose;         // Duration of the
       deceleration phase on the closing process [0..15sec].
    unsigned char motorPower;                // Motor Strength level
       [1..9].
    unsigned char motorSensitivity;          // Motor current
       sensitivity level during \textit{full} speed moving [0..9].
    unsigned char \textit{walk}Time;                  // Time to open
       the gate for peons passing [0..99].
    unsigned char autoTime\textit{full}Close;        // Pause time
       after opening process that the system waits until it starts the
        close process automatically [0..99sec, being 0-OFF].
    unsigned char autoTime\textit{walk}Close;        // Pause time
       after peons passage opening process that the system waits until
        it starts the close process automatically [0..99sec, being 0-
       OFF].
    unsigned char photoCellIsON;             // Photo cell state [0-OFF
       , 1-ON].
    unsigned char photoCellInOpen;           // Photo cell used on the
       opening process [0-Not used, 1-Used].
    unsigned char securityBandIsON;          // Security Band state [0-
       Not installed, 1-Installed and used].
    unsigned char securityBandType;          // Security Band Type [0-8
       K2 type, 1-NC type].
    unsigned char securityBandInOpen;        // Security Band used on
       the opening process [0-Not used, 1-Used].
    unsigned char operationMode;             // Operation mode [0-Modo
       autom tico, 1-Modo passo-a-passo, 2-Modo Condom nio].
    unsigned char flashLightMode;            // Flashlight operation
       mode [0-1Hz Blink closing, 0.5Hz openning and ON at autoTime
       ????Close time; 1- ON opening and closing and OFF at autoTime
       ????Close time; 2- ON opening, closing, at autoTime????Close
       time and ligthTime minutes after closing]. (rev1.149)
    unsigned char programmingDistance;       // Distance programming
       enable [0-Disabled, 1-Enabled].
    unsigned char decelarationSensivity;     // Motor current
       sensitivity level during deceleration moving phase [0..9].


    //page1
    unsigned char homemPresente;             // Indicates if the gate
       is controlled only manually by a person [0-Controlled normally,
        1-Controlled only by hand on the buttons].
    unsigned char logicDigital;              // Defines the buttons
       functioning logic [0- Button 1 is \textit{full} open and close,
        Button 2 is peons open and close; 1- Button 1 is \textit{full}
```

```
    open and Button 2 is \textit{full} close].
unsigned char softStart;                // Soft start enable state
    [0-Disabled, 1-Enabled].
unsigned char softstop;                 // Soft sto enable state
   [0-Disabled, 1-Enabled].
unsigned char ligthTime;                // Time for the flashlight
    ON after closing the gate (implies flashLightMode=2) [0..99
   minutes].
unsigned char folow_me;                 // Indicates if the follow
    me mode is active. When active, the gate initiates the close
   process 3 seconds after a passage trough the photocell. (rev1
   .149)
unsigned char Stopboton;                // Activa o stop botton
unsigned char electricBrake;            // Electric brake state
   [0-Disabled, 1-Enabled].
unsigned char velocityDecelaration;     // Velocity to decelerate
   the gate in levels [1..9].
unsigned char flashRGBMode;             // Indicates the RGB
   flashlight functioning mode [0-Continuous mode, 1-0.5Hz
   Blinking mode].
unsigned char reserved10;         // Difene the type of the
   deceration us want. The decelation have two type one have 2
   semi-cles and more 1 semi-cicle and other have only one semi
   cycle one and other cycle off.     unsigned char reserved3;
unsigned char Direction_motor;          // Difene the direcion of
   motor. Can choose the direccion and change the limitswitch and
   motor relay.
unsigned char TypeofMotor;              // Escolhe o tipo de motor
    que ir  ser usado.
unsigned char reserved;                 // Indicates when the
   motor has a movement encoder to control his movement.
unsigned char reserved6;
unsigned char reserved7;
//unsigned char reserved8;


//page2
unsigned char positionRemotes\textit{full};
unsigned char positionRemotes\textit{walk};
unsigned long counterMoves;

unsigned char OnlyRollingCode;
unsigned char reserved12;
unsigned long learningCurrentDecelarationClose;
unsigned long learningCurrentDecelarationOpen;


//page3
unsigned int learningCurrentNormalClose;
unsigned int learningCurrentNormalOpen;
unsigned long learningTimeToOpen;
unsigned long learningTimeToClose;
unsigned char reserved31;
unsigned char reserved32;
unsigned char reserved33;
unsigned char reserved34;
```

```
}varSystem_NVM;
```

Por outro lado, a segunda estrutura contém parâmetros apenas de leitura que mão se prevê serem alterados.

```
typedef struct
{
        stateMotor_enum StateMotor;
    StateEnum       photoCellIsObstructed;
    StateEnum       SecurityBarIsObstructed;
    StateEnum       FimCurso_CloseIsEnabled;
    StateEnum       FimCurso_OpenIsEnabled;
    int             decelerationOpenCurrent;
    int             decelerationCloseCurrent;
    char            StartFromButton;
    unsigned char   WaitTimeCloseInitial;
    stateMotor_enum LastState;
    StateEnum       TriacON;
    StateEnum       InStoping;
    int             velocityFactor;
    int             velocityFactorstop;
    char            AutoInversionActiveStop;
    char            SoftStopDecrementControl;
    char            SoftStartDecrementControl;
    char            Torquerelanty;
    char            Activatecounter;
    char            Counter_Learning;
    unsigned long   TimeMaxMotorIsON;               // Maximum allowed
        ON time for the motor.
    char            Statedoorcontrol;
    char            preflashingcontrol;

    signed long     PositionActual;
    StateEnum       DigitSinalizedTemp;
    unsigned char   AutoCloseActive;
    char            upFaseRGB;
    char            upFaseFlashligth;
    char            LearningIsEnabled;
    char            AutoInversionActive;

    char            ControlReleCapacitorOpen;
    char            ControlReleCapacitorClose;
    char            programinAutomatic;

    StateEnum       photoCellMakeErrorOpen;
    StateEnum       photoCellMakeErrorClose;
    StateEnum       SecurityBarMakeError;
    unsigned int    ActualCurrent;
    unsigned long   ActualHistCurrent;
    unsigned long   ActualComparatorCurrent;
    int             velocityActual;
    StateEnum       StateFollowIsOn;
    TypeCMD         ProgrammingDistanceIs;
    unsigned long   actualCounterMoves;
    StateEnum       DistanceProgrammingActive;//Serve Para Sinalizar a
        FlashLigth
    StateEnum       CurrentAlarmIsOn;
```

```
    AlarmesStateEnum        WorkTimeMaxAlarmState;
    char              NumberOffErrors;
    StateEnum         InvertionCurrentClosing;
    StateEnum         InvertionCurrentOpening;
    StateEnum         InvertionClosingFromOpen;
    StateEnum         LearningDecelaration;
    StateEnum         PositionIsLost;
    char              StateVersion;
    char              Time\textit{walk}isactived;



    unsigned int      ADCZeroOffset;
    unsigned char         Control50or60hz;
    unsigned char   frequenciamotor;

    char              showAP;
    char              passoAPassoAutoClose;

    #ifdef   TEST_RESET
    char              showReset;
    #endif

}varSystem;
```

De ambas estas listas foram retirados os parâmetros principais e listados na tabela da página seguinte.

2 Endereçamento dos parâmetros

Com os parâmetros relevantes listados, atribuiu-se um endereço virtual que permite identifica-los e manipula-los. Os endereços ocupam 1 byte e endereçam 2 bytes permitindo endereçar um bloco de memória até 512 bytes. Os endereços de memória atribuídos constam nas colunas azuis e os respetivos índices (endereçáveis ao byte) constam nas colunas verdes. As ultimas duas colunas contém variáveis utilizadas em sistemas precedentes e que devem ser suportadas pelo sistema. Estas variáveis possuem endereços redundantes, podendo ser acedidos, quer através do seu endereço antigo quer do novo, permitindo a portabilidade. Os parâmetros foram endereçados sequencialmente a partir do endereço livre mais baixo (0x10) e agrupados por natureza de conteúdo (as variáveis com conteúdo semelhante e com 1 byte são agrupadas no mesmo endereço).

					for compatibility	
origem	size (bytes)	variable name	primary address	address index	secundary address	address index
	1	decelarationOpen	10	1	0	1
	1	decelarationClose		0	0	0
	1	motorPower	11	1	~~	~~
	1	motorSensitivity		0	~~	~~
	1	walkTime	12	0	~~	~~
	1	autoTimeFullClose	13	1	2	0
	1	autoTimeWalkClose		0	~~	~~
	1	photoCellIsON	14	1	5	0
	1	photoCellInOpen		0	~~	~~
	1	securityBandIsON	15	1	~~	~~
	1	securityBandType		0	~~	~~
	1	securityBandInOpen	16	0	~~	~~
	1	operationMode	17	0	~~	~~
	1	flashLightMode	18	0	~~	~~
	1	programmingDistance	19	0	~~	~~
	1	decelarationSensivity	1A	0	~~	~~
	1	homemPresente	1B	0	~~	~~
	1	logicDigital	1C	0	7	0
	1	softStart	1D	1	~~	~~
	1	softstop		0	~~	~~
	1	ligthTime	1E	0	3	0
	1	folow_me	1F	0	A	0
	1	Stopboton	20	0	~~	~~
	1	electricBrake	21	0	~~	~~
	1	velocityDecelaration	22	0	~~	~~
	1	flashRGBMode	23	0	8	0
	1	Direction_motor	24	0	~~	~~
	1	TypeofMotor	25	0	~~	~~
	1	positionRemotesFull	26	1	~~	~~
	1	positionRemotesWalk		0	~~	~~
	4	counterMoves	27-28	~~	~~	~~
	1	OnlyRollingCode	29	0	~~	~~
	4	learningCurrentDecelarationClose	2A-2B	~~	~~	~~
	4	learningCurrentDecelarationOpen	2C-2D	~~	~~	~~

					for compatibility	
origem	size (bytes)	variable name	primary address	address index	secundary address	address index
	2	learningCurrentNormalClose	2E	~~	~~	~~
	2	learningCurrentNormalOpen	2F	~~	~~	~~
	4	learningTimeToOpen	30-31	~~	~~	~~
	4	learningTimeToClose	32-33	~~	~~	~~
inputs.h	1	RFFull	34	0		
varSystem	1	photoCellIsObstructed	35	1		
	1	SecurityBarIsObstructed	35	0		
	1	FimCurso_CloseIsEnabled	36	1		
	1	FimCurso_OpenIsEnabled	36	0		
	1	Statedoorcontrol	37	0		
	4	PositionActual	38-39	~~		
	1	PositionIsLost	3A	1		
	1	StateVersion	3A	0		

Figure 1: Mapeamento de endereços

3 Comunicação por *uart*

A comunicação entre o *pic18f46k40* e o *esp 32 c3* é feita através de porta serie com *baudrate* 9600, *data size* de 8 *bits* com *start* e *stop bits* e sem controlo nem verificação de trama. As portas utilizadas foram a *eusart* 1 do *pic18f46k40* e a *uart* 1 do *esp 32 c3* (mapeado em *GPIO* 18 e *GPIO* 19).

- *Baudrate*: 9600
- *Data size*: 8 *bits*
- *Start bit*: 1 *bit*
- *Stop bit*: 1 *bit*
- *Parity*: *None*
- *Flow control*: *None*

startbit	data[7]	data[6]	data[5]	data[4]	data[3]	data[2]	data[1]	data[0]	stopbit

```
/* Configure parameters of an UART driver,
 * communication pins and install the driver */
uart_config_t uart_config = {
    .baud_rate = 9600,
    .data_bits = UART_DATA_8_BITS,
    .parity = UART_PARITY_DISABLE,
    .stop_bits = UART_STOP_BITS_1,
    .flow_ctrl = UART_HW_FLOWCTRL_DISABLE,
    .source_clk = UART_SCLK_APB,
};
//Install UART driver, and get the queue.
uart_driver_install(EX_UART_NUM, BUF_SIZE * 2, BUF_SIZE * 2, 20, &
   uart1_queue, 0);
uart_param_config(EX_UART_NUM, &uart_config);
//Set UART pins (using UART0 default pins ie no changes.)
uart_set_pin(EX_UART_NUM, 19, 18, UART_PIN_NO_CHANGE,
   UART_PIN_NO_CHANGE);
```

4 *Package*

Para estruturar a comunicação dos dados é utilizada uma estrutura que implementa um *package*. Um *package* é composto por um *start byte*, um *function code* (1 *byte*), um *address* (1 *byte*), um *data block* (2 *bytes*) e um *end byte*. O *start byte* e *stop byte* convencionais são 0x0A e 0x0D respetivamente. Estes asseguram que os comandos são devidamente recebidos sem perda de bytes e marcam o inicio e fim de um *package*. O *function code* descreve o tipo de função que o *package* pretende transmitir e os seus valores possíveis estão enumerados na lista seguinte.

```
enum functioncode_t{
    READ=0,
    WRITE=1,
    PROGRAMMING_ENABLE=2,
    CONFIRM=3,
    NUM_COMMANDS_F=4,
    NUM_EMPTY_COMMANDS_F=5,
    OCCUPIED_POS_F=6,
    EMPTY_POS_F=7,
    SAVE_COMMAND_F=8,
    ERASE_COMMAND_F=9,
```

```
    READ_SERIAL_F=10,
    NUM_COMMANDS_W=11,
    NUM_EMPTY_COMMANDS_W=12,
    OCCUPIED_POS_W=13,
    EMPTY_POS_W=14,
    SAVE_COMMAND_W=15,
    ERASE_COMMAND_W=16,
    READ_SERIAL_W=17,
    READ_ALL=18
};
```

O *address* indica o endereço virtual de 2 *bytes* onde os dados que se pretende mencionar estão armazenados. O *data block* contém o valor dos dados (também pode conter valores de retorno dependendo do *function code*).

Inclusivamente o *package* contém uma *flag valid* que identifica se a *package* está válida para ser enviada ou se foi recebida com uma falha.

```
struct package_t{
    //general variables
    enum functioncode_t functioncode;
    uint8_t address;
    union packagedata_t data;
    //control variables
    uint8_t startbyte;
    uint8_t endbyte;
    bool valid;
};
```

5 Funções

As funções suportadas pelo *parser* do sistema foram tabeladas e implementadas a partir da tabela abaixo.

	Function List								
	function number	name	description	structure					done
				start byte	function code	address	data	end byte	
	0	read	allows to read from the memory	0x0A	0x00	reading address	retrieved data	0x0D	1
	1	write	allows to write into the memory	0x0A	0x01	writing address	written data	0x0D	1
	2	programming enable	enables/saves memory changes	0x0A	0x02	0x00	enabled(1)/disabled(0)/blocked(2)	0x0D	1
	3	confirm	confirms a previous command	0x0A	0x03	0x00	success(1)/fail(0)	0x0D	1
Full	4	num commands	returns the number of saved commands	0x0A	0x04	0x00	nº commands	0x0D	1
Full	5	num empty commands	returns the number of empty spots for commands	0x0A	0x05	0x00	nº empty pos	0x0D	1
Full	6	occupied pos	returns the positions occupied by commands	0x0A	0x06	relative pos	absolute pos	0x0D	1
Full	7	empty pos	returns the empty positions for commands	0x0A	0x07	relative pos	absolute pos	0x0D	1
Full	8	save command	saves a command (relative pos)	0x0A	0x08	relative pos	serial (1st half then 2nd half)	0x0D	1
Full	9	erase command	erases a command (relative pos)	0x0A	0x09	relative pos	0x0000	0x0D	1
Full	10	read serial	read the serial of a saved command	0x0A	0x0A	relative pos	serial (1st half then 2nd half)	0x0D	1
Walk	11	num commands	returns the number of saved commands	0x0A	0x0B	0x00	nº commands	0x0D	1
Walk	12	num empty commands	returns the number of empty spots for commands	0x0A	0x0C	0x00	nº empty pos	0x0D	1
Walk	13	occupied pos	returns the positions occupied by commands	0x0A	0x0D	relative pos	absolute pos	0x0D	1
Walk	14	empty pos	returns the empty positions for commands	0x0A	0x0E	relative pos	absolute pos	0x0D	1
Walk	15	save command	saves a command (relative pos)	0x0A	0x0F	relative pos	serial (1st half then 2nd half)	0x0D	1
Walk	16	erase command	erases a command (relative pos)	0x0A	0x10	relative pos	0x0000	0x0D	1
Walk	17	read serial	read the serial of a saved command	0x0A	0x11	relative pos	serial (1st half then 2nd half)	0x0D	1
	18	read all	reads all the readable parameters	0x0A	0x12	0x00	0x0000	0x0D	1

Figure 2: Mapeamento de endereços

5.1 *READ*

A função *READ* é responsável por retornar o conteúdo presente dentro de uma posição virtual da memória. O parâmetro *function code* é 0x00. No parâmetro *address* é indicado o endereço que se pretende ler de entre a lista apresentada na tabela 2. No parâmetro *data block* é devolvido o valor da

variável contida dentro do endereço indicado. Deve ser devolvido um comando de confirmação após a execução.

5.2 *WRITE*

A função *WRITE* é responsável por alterar um parâmetro especificado numa determinada posição de memória. O parâmetro *function code* é 0x01. No parâmetro *address* é indicado o endereço da variável que se pretende modificar de entre a lista apresentada na tabela 2. No parâmetro *data block* é indicado o novo valor da variável indicada. Deve ser devolvido um comando de confirmação após a execução.

5.3 *PROGRAMMING ENABLE*

A função *PROGRAMMING ENABLE* é responsável por alternar entre o modo programação. O modo programação só pode ser ativado em estado *standby* e só permite o funcionamento do sistema após ser desativado. Enquanto está desativado, as funções de escrita e de mudança de comportamento do sistema são bloqueadas. O parâmetro *function code* é 0x02. O parâmetro *address* é 0x00. No parâmetro *data block* é indicado o estado do *programming mode*, inativo (0x00), ativo (0x01) ou bloqueado pelo sistema(0x02). Deve ser devolvido um comando de confirmação após a execução.

5.4 *CONFIRM*

A função *CONFIRM* confirma a realização de um comando anterior. Esta confirmação pode ser afirmativa ou negativa. O parâmetro *function code* é 0x03. O parâmetro *address* é 0x00. No parâmetro *data block* é indicado o tipo de confirmação, falha (0x00) ou sucesso (0x01). Pode ser devolvido um comando de confirmação após a execução (para confirmar a receção).

5.5 *NUM COMMANDS F*

A função *NUM COMMANDS F* indica o numero de comandos distintos do tipo *full* registados no sistema. O parâmetro *function code* é 0x04. O parâmetro *address* é 0x00. No parâmetro *data block* é indicado o numero de comandos do tipo *full* registados no sistema (até um máximo de 99). Deve ser devolvido um comando de confirmação após a execução.

5.6 *NUM EMPTY COMMANDS F*

A função *NUM EMPTY COMMANDS F* indica o numero de espaços vazios no sistema para comandos distintos do tipo *full*. O parâmetro *function code* é 0x05. O parâmetro *address* é 0x00. No parâmetro *data block* é indicado o numero de espaços vazios no sistema para comandos distintos do tipo *full* (até um máximo de 99). Deve ser devolvido um comando de confirmação após a execução.

5.7 *OCCUPIED POS F*

A função *OCCUPIED POS F* indica os espaços da lista de comandos ocupados por comandos distintos do tipo *full*. O parâmetro *function code* é 0x06. No parâmetro *address* é indicada a ordem relativa do comando em relação ao inicio da lista. No parâmetro *data block* é indicada a posição absoluta na mesma lista. Na ocorrência de mais de um comando estar registado esta função deve ser enviada por cada ocorrência. Deve ser devolvido um comando de confirmação após a execução de todas as funções.

5.8 *EMPTY POS F*

A função *EMPTY POS F* indica os espaços livres da lista de comandos do tipo *full*. O parâmetro *function code* é 0x07. No parâmetro *address* é indicada a ordem relativa do espaço vazio em relação ao inicio da lista. No parâmetro *data block* é indicada a posição absoluta na mesma lista. Na ocorrência de existir mais do que um espaço livre esta função deve ser enviada por cada ocorrência. Deve ser devolvido um comando de confirmação após a execução de todas as funções.

5.9 *SAVE COMMAND F*

A função *SAVE COMMAND F* regista um novo comando na lista de comandos do tipo *full*. O parâmetro *function code* é 0x08. No parâmetro *address* é indicada a ordem relativa do espaço vazio em relação ao inicio da lista onde se pretende registar o comando. No parâmetro *data block* é indicado o numero de serie do comando. Como o numero de serie tem o dobro do tamanho do parâmetro *data block*, o numero de serie deve ser enviado em dois *packages* consecutivos (no caso de interrupção após o primeiro package a função é cancelada). Deve ser devolvido um comando de confirmação após a execução de todas as funções.

5.10 *ERASE COMMAND F*

A função *ERASE COMMAND F* elimina um comando na lista de comandos do tipo *full*. O parâmetro *function code* é 0x09. No parâmetro *address* é indicada a ordem relativa do comando em relação ao inicio da lista de onde se pretende eliminar o comando. O parâmetro *data block* é 0x0000. Deve ser devolvido um comando de confirmação após a execução.

5.11 *READ SERIAL F*

A função *READ SERIAL F* devolve o numero de serie de um comando da lista de comandos do tipo *full*. O parâmetro *function code* é 0x0A. No parâmetro *address* é indicada a ordem relativa do comando em relação ao inicio da lista de onde se pretende ler o valor serie. No parâmetro *data block* é devolvida uma das metades do valor serie do comando (esta função retorna o valor serie em 2 packages). Deve ser devolvido um comando de confirmação após a execução de todas as funções.

5.12 *NUM COMMANDS W*

A função *NUM COMMANDS W* indica o numero de comandos distintos do tipo *walk* registados no sistema. O parâmetro *function code* é 0x0B. O parâmetro *address* é 0x00. No parâmetro *data block* é indicado o numero de comandos do tipo *walk* registados no sistema (até um máximo de 99). Deve ser devolvido um comando de confirmação após a execução.

5.13 *NUM EMPTY COMMANDS W*

A função *NUM EMPTY COMMANDS W* indica o numero de espaços vazios no sistema para comandos distintos do tipo *walk*. O parâmetro *function code* é 0x0C. O parâmetro *address* é 0x00. No parâmetro *data block* é indicado o numero de espaços vazios no sistema para comandos distintos do tipo *walk* (até um máximo de 99). Deve ser devolvido um comando de confirmação após a execução.

5.14 *OCCUPIED POS W*

A função *OCCUPIED POS W* indica os espaços da lista de comandos ocupados por comandos distintos do tipo *walk*. O parâmetro *function code* é 0x0D. No parâmetro *address* é indicada a ordem relativa do comando em relação ao inicio da lista. No parâmetro *data block* é indicada a posição absoluta na mesma lista. Na ocorrência de mais de um comando estar registado esta função deve ser enviada por cada ocorrência. Deve ser devolvido um comando de confirmação após a execução de todas as funções.

5.15 *EMPTY POS W*

A função *EMPTY POS W* indica os espaços livres da lista de comandos do tipo *walk*. O parâmetro *function code* é 0x0E. No parâmetro *address* é indicada a ordem relativa do espaço vazio em relação ao inicio da lista. No parâmetro *data block* é indicada a posição absoluta na mesma lista. Na ocorrência de existir mais do que um espaço livre esta função deve ser enviada por cada ocorrência. Deve ser devolvido um comando de confirmação após a execução de todas as funções.

5.16 *SAVE COMMAND W*

A função *SAVE COMMAND W* regista um novo comando na lista de comandos do tipo *walk*. O parâmetro *function code* é 0x0F. No parâmetro *address* é indicada a ordem relativa do espaço vazio em relação ao inicio da lista onde se pretende registar o comando. No parâmetro *data block* é indicado o numero de serie do comando. Como o numero de serie tem o dobro do tamanho do parâmetro *data block*, o numero de serie deve ser enviado em dois *packages* consecutivos (no caso de interrupção após o primeiro *package* a função é cancelada). Deve ser devolvido um comando de confirmação após a execução de todas as funções.

5.17 *ERASE COMMAND W*

A função *ERASE COMMAND W* elimina um comando na lista de comandos do tipo *walk*. O parâmetro *function code* é 0x10. No parâmetro *address* é indicada a ordem relativa do comando em relação ao inicio da lista de onde se pretende eliminar o comando. O parâmetro *data block* é 0x0000. Deve ser devolvido um comando de confirmação após a execução.

5.18 *READ SERIAL W*

A função *READ SERIAL W* devolve o numero de serie de um comando da lista de comandos do tipo *walk*. O parâmetro *function code* é 0x11. No parâmetro *address* é indicada a ordem relativa do comando em relação ao inicio da lista de onde se pretende ler o valor serie. No parâmetro *data block* é devolvida uma das metades do valor serie do comando (esta função retorna o valor serie em 2 packages). Deve ser devolvido um comando de confirmação após a execução de todas as funções.

5.19 *READ ALL*

A função *READ ALL* devolve a leitura de todos os endereços abrangidos pelo comando *READ*. É utilizado para encurtar o numero de *packages* de leitura requisitados. O parâmetro *function code* é 0x12. O parâmetro *address* é 0x00. O parâmetro *data block* é 0x0000. Deve ser devolvido um comando de confirmação após a execução de todas as funções.

6 Receção assíncrona de *bytes*

Os bytes enviados através da porta uart entre o *pic18f46k40* e o *esp 32 c3* são recebidos assincronamente, sendo necessário encapsular os bytes em *packages*. Para implementar esta funcionalidade desenvolveu-se um *package buffer* que permite o armazenamento de até 16 *packages* num *array* circular. O *buffer* é controlado por 3 apontadores, um controla o *byte* de escrita, outro controla o *package* de escrita e o último controla o *package* de leitura. Os bytes são armazenados dentro do *buffer* em ordem sequencial até preencher um *package*, caso o *package* esteja válido é disponibilizado para leitura, caso contrário é descartado.

```
#define BUFFER_SIZE 16
struct packagebuffer_t{
        struct package_t buffer[BUFFER_SIZE];
        uint8_t writeByteIndex;
        uint8_t writePackageIndex;
        uint8_t readPackageIndex;
};
```

7 *Strings .json*

Para implementar a conversão de dados provenientes de *packages* em ficheiros *.json* foi necessário desenhar uma biblioteca que formatasse os dados. Um ficheiro *.json* organiza os dados em *JavaScript Object Notation*, e troca-os de modo simples e rápido entre sistemas. Os dados estão contidos dentro de *tokens* e organizam-se nos tipos listados abaixo.

```
enum jsontype_t {
    JSON_NULL = -1,
    JSON_PRIMITIVE = 0,
    JSON_OBJECT = 1,
    JSON_ARRAY = 2,
    JSON_STRING = 3,
    JSON_INTEGER = 4,
    JSON_FLOAT = 5
};
```

O tipo *JSON NULL* representa um *token* inválido e o tipo *JSON PRIMITIVE* divide-se num dos seguintes estados.

```
enum primitivetype_t {
    PRIMITIVE_FALSE = 0,
    PRIMITIVE_TRUE = 1,
    PRIMITIVE_NULL = 2
};
```

Note-se que o tipo de *token JSON ARRAY* não foi completamente implementado.

Para gerar uma *string* em formato *.json* foi necessário inicializar um *scope* vazio. Esse *scope* representa um ficheiro sem tokens associados.

```
{
}
```

Os *tokens* são compostos por um nome e um valor. O nome está compreendido entre aspas, enquanto que os valores veriam conforme o seu tipo.

- O tipo primitive é uma keyword entre aspas.
- O tipo object é um scope com tokens no seu interior.
- O tipo array é limitado por parenteies quadrados.
- O tipo string é limitado por aspas
- O tipo integer é um numero sem ponto decimal
- O tipo integer é um numero com ponto decimal

Para adicionar um *token* a uma string *.json* é necessário indicar o nome e valor do *token*. Por outro lado, é possível retirar valores de uma string *.json* seguindo o processo oposto.