Rafael Lemos Pedro

Comunicação serie pic18f46k40 - esp32c3

September 8, 2022

Contents

1 Dados parameterizáveis

Os dados parametrizáveis relevantes em ambas as centrais *pic18f46k40* estão instanciados dentro de duas estruturas. Estas estruturas são compostas por 4 parâmetros, endereço da variável, subendereço virtual (0-1), endereço virtual (0x00-0xFF) e tamanho em bits (B8, B16 ou B32).

1.1 MC50UNI V.0.2.3.X

```
struct Parameter systemVarAddresses[]={
    {&var_sys_NVM.decelarationOpen, 1, 0x10, B8},
    {&var_sys_NVM.decelarationClose, 0, 0x10, B8},
    {&var_sys_NVM.motorPower, 1, 0x11, B8},
    {&var_sys_NVM.motorSensitivity, 0, 0x11, B8},
    {&var_sys_NVM.walkTime, 0, 0x12, B8},
    {&var_sys_NVM.autoTimeFullClose, 1, 0x13, B8},
    {&var_sys_NVM.autoTimeWalkClose, 0, 0x13, B8},
    {&var_sys_NVM.photoCellIsON, 1, 0x14, B8},
    {&var_sys_NVM.photoCellInOpen, 0, 0x14, B8},
    {&var_sys_NVM.securityBandIsON, 1, 0x15, B8},
    {&var_sys_NVM.securityBandType, 0, 0x15, B8},
    {&var_sys_NVM.securityBandInOpen, 0, 0x16, B8},
    {&var_sys_NVM.operationMode, 0, 0x17, B8},
    {&var_sys_NVM.flashLightMode, 0, 0x18, B8},
    {&var_sys_NVM.programmingDistance, 0, 0x19, B8},
    {&var_sys_NVM.decelarationSensivity, 0, 0x1A, B8},

    {&var_sys_NVM.homemPresente, 0, 0x1B, B8},
    {&var_sys_NVM.logicDigital, 0, 0x1C, B8},
    {&var_sys_NVM.softStart, 1, 0x1D, B8},
    {&var_sys_NVM.softstop, 0, 0x1D, B8},
    {&var_sys_NVM.ligthTime, 0, 0x1E, B8},
    {&var_sys_NVM.folow_me, 0, 0x1F, B8},
    {&var_sys_NVM.Stopboton, 0, 0x20, B8},
    {&var_sys_NVM.electricBrake, 0, 0x21, B8},
    {&var_sys_NVM.velocityDecelaration, 0, 0x22, B8},
    {&var_sys_NVM.flashRGBMode, 0, 0x23, B8},
    {&var_sys_NVM.Direction_motor, 0, 0x24, B8},
    {&var_sys_NVM.TypeofMotor, 0, 0x25, B8},

    {&var_sys_NVM.positionRemotesFull, 1, 0x26, B8},
    {&var_sys_NVM.positionRemotesWalk, 0, 0x26, B8},
    {&var_sys_NVM.counterMoves, 0, 0x27, B32},
    {&var_sys_NVM.OnlyRollingCode, 0, 0x29, B8},
    {&var_sys_NVM.learningCurrentDecelarationClose, 0, 0x2A, B32},
    {&var_sys_NVM.learningCurrentDecelarationOpen, 0, 0x2C, B32},

    {&var_sys_NVM.learningCurrentNormalClose, 0, 0x2E, B16},
    {&var_sys_NVM.learningCurrentNormalOpen, 0, 0x2F, B16},
    {&var_sys_NVM.learningTimeToOpen, 0, 0x30, B32},
    {&var_sys_NVM.learningTimeToClose, 0, 0x32, B32},

    {&var_sys.RFFull, 0, 0x34, B8},

    {&var_sys.photoCellIsObstructed, 1, 0x35, B8},
    {&var_sys.SecurityBarIsObstructed, 0, 0x35, B8},
    {&var_sys.FimCurso_CloseIsEnabled, 1, 0x36, B8},
```

```
    {&var_sys.FimCurso_OpenIsEnabled, 0, 0x36, B8},
    {&var_sys.Statedoorcontrol, 0, 0x37, B8},
    {&var_sys.PositionActual, 0, 0x38, B32},
    {&var_sys.PositionIsLost, 1, 0x3A, B8},
    {&var_sys.StateVersion, 0, 0x3A, B8},
};
```

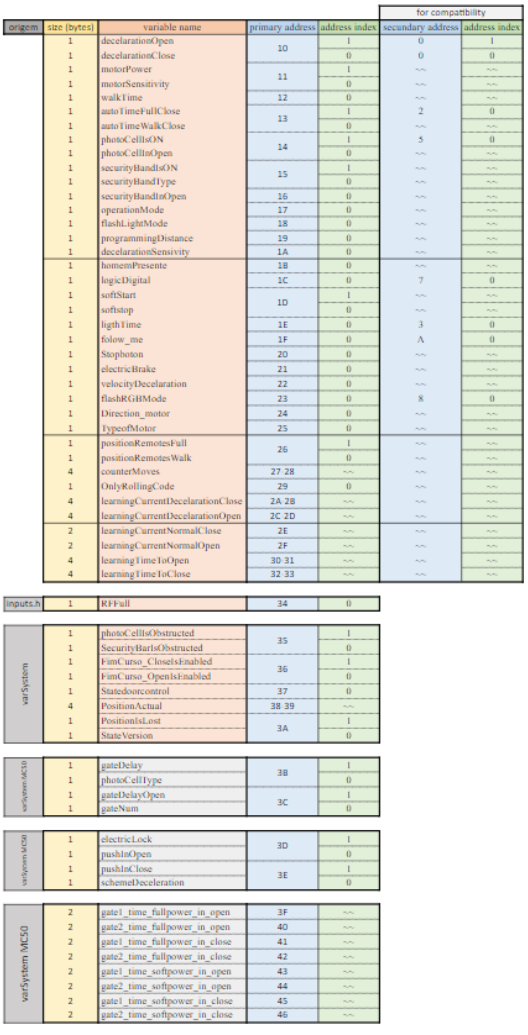
1.2 MC52 V.1.3.0

```
struct Parameter systemVarAddresses[]={
    {&var_sys_NVM.motorPower, 1, 0x11, B8},
    {&var_sys_NVM.walkTime, 0, 0x12, B8},
    {&var_sys_NVM.autoTimeFullClose, 1, 0x13, B8},
    {&var_sys_NVM.autoTimeWalkClose, 0, 0x13, B8},
    {&var_sys_NVM.photoCellIsON, 1, 0x14, B8},
    {&var_sys_NVM.photoCellInOpen, 0, 0x14, B8},
    {&var_sys_NVM.securityBandIsON, 1, 0x15, B8},
    {&var_sys_NVM.securityBandType, 0, 0x15, B8},
    {&var_sys_NVM.securityBandInOpen, 0, 0x16, B8},
    {&var_sys_NVM.operationMode, 0, 0x17, B8},
    {&var_sys_NVM.flashLightMode, 0, 0x18, B8},
    {&var_sys_NVM.programmingDistance, 0, 0x19, B8},

    {&var_sys_NVM.homemPresente, 0, 0x1B, B8},
    {&var_sys_NVM.logicDigital, 0, 0x1C, B8},
    {&var_sys_NVM.softStart, 1, 0x1D, B8},
    {&var_sys_NVM.ligthTime, 0, 0x1E, B8},
    {&var_sys_NVM.folow_me, 0, 0x1F, B8},
    {&var_sys_NVM.electricBrake, 0, 0x21, B8},
    {&var_sys_NVM.velocityDecelaration, 0, 0x22, B8},
    {&var_sys_NVM.flashRGBMode, 0, 0x23, B8},

    {&var_sys_NVM.positionRemotesFull, 1, 0x26, B8},
    {&var_sys_NVM.positionRemotesWalk, 0, 0x26, B8},
    {&var_sys_NVM.counterMoves, 0, 0x27, B32},
    {&var_sys_NVM.OnlyRollingCode, 0, 0x29, B8},

    {&var_sys_NVM.gateDelay, 1, 0x3B, B8},
    {&var_sys_NVM.photoCellType, 0, 0x3B, B8},
    {&var_sys_NVM.gateDelayOpen, 1, 0x3C, B8},
    {&var_sys_NVM.gateNum, 0, 0x3C, B8},

    {&var_sys_NVM.electricLock, 1, 0x3D, B8},
    {&var_sys_NVM.pushInOpen, 0, 0x3D, B8},
    {&var_sys_NVM.pushInClose, 1, 0x3E, B8},
    {&var_sys_NVM.schemeDeceleration, 0, 0x3E, B8},

    {&var_sys_NVM.gate1_time_fullpower_in_open, 0, 0x3F, B16},
    {&var_sys_NVM.gate2_time_fullpower_in_open, 0, 0x40, B16},
    {&var_sys_NVM.gate1_time_fullpower_in_close, 0, 0x41, B16},
    {&var_sys_NVM.gate2_time_fullpower_in_close, 0, 0x42, B16},
    {&var_sys_NVM.gate1_time_softpower_in_open, 0, 0x43, B16},
    {&var_sys_NVM.gate2_time_softpower_in_open, 0, 0x44, B16},
    {&var_sys_NVM.gate1_time_softpower_in_close, 0, 0x45, B16},
    {&var_sys_NVM.gate2_time_softpower_in_close, 0, 0x46, B16},
};
```

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

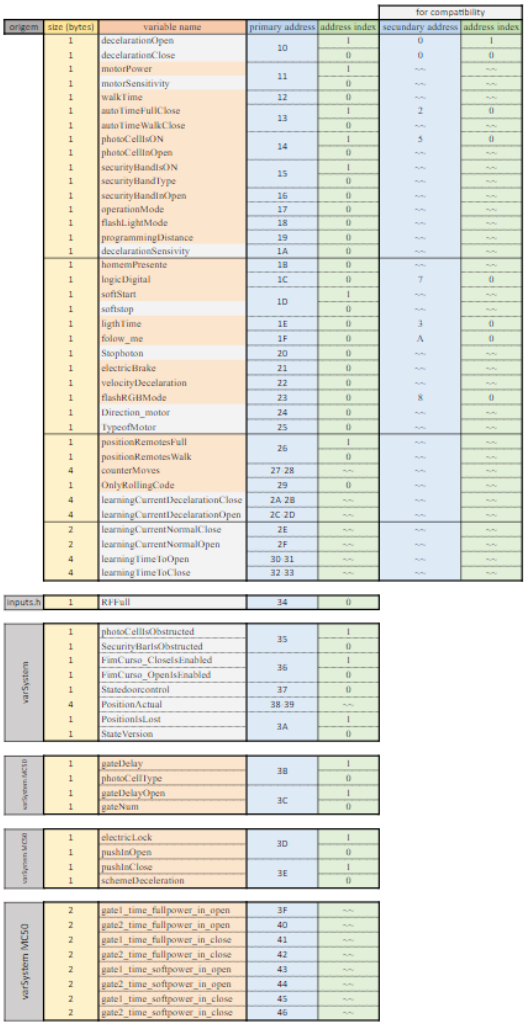
2 Endereçamento dos parâmetros

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). Os parâmetros comuns entre centrais estão registados no mesmo endereço de modo a garantir compatibilidade



origem	size (bytes)	variable name	primary address	address index	for compatibility: secundary address	for compatibility: 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	~~	~~	~~
	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		0		
	1	FimCurso_CloseIsEnabled	36	1		
	1	FimCurso_OpenIsEnabled		0		
	1	Statedoorcontrol	37	0		
	4	PositionActual	38 39	~~		
	1	PositionIsLost	3A	1		
	1	StateVersion		0		
varSystem MC16	1	gateDelay	3B	1		
	1	photoCellType		0		
	1	gateDelayOpen	3C	1		
	1	gateNum		0		
varSystem MC50	1	electricLock	3D	1		
	1	pushInOpen		0		
	1	pushInClose	3E	1		
	1	schemeDeceleration		0		
varSystem MC50	2	gate1_time_fullpower_in_open	3F	~~		
	2	gate2_time_fullpower_in_open	40	~~		
	2	gate1_time_fullpower_in_close	41	~~		
	2	gate2_time_fullpower_in_close	42	~~		
	2	gate1_time_softpower_in_open	43	~~		
	2	gate2_time_softpower_in_open	44	~~		
	2	gate1_time_softpower_in_close	45	~~		
	2	gate2_time_softpower_in_close	46	~~		

Figure 1: Mapeamento de endereços MC50UNI V.0.2.3.X



origem	size (bytes)	variable name	primary address	address index	for compatibility: secundary address	for compatibility: 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	~~	~~	~~
	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		0		
	1	FimCurso_CloseIsEnabled	36	1		
	1	FimCurso_OpenIsEnabled		0		
	1	Statedoorcontrol	37	0		
	4	PositionActual	38 39	~~		
	1	PositionIsLost	3A	1		
	1	StateVersion		0		
varSystem MC16	1	gateDelay	3B	1		
	1	photoCellType		0		
	1	gateDelayOpen	3C	1		
	1	gateNum		0		
varSystem MC50	1	electricLock	3D	1		
	1	pushInOpen		0		
	1	pushInClose	3E	1		
	1	schemeDeceleration		0		
varSystem MC50	2	gate1_time_fullpower_in_open	3F	~~		
	2	gate2_time_fullpower_in_open	40	~~		
	2	gate1_time_fullpower_in_close	41	~~		
	2	gate2_time_fullpower_in_close	42	~~		
	2	gate1_time_softpower_in_open	43	~~		
	2	gate2_time_softpower_in_open	44	~~		
	2	gate1_time_softpower_in_close	45	~~		
	2	gate2_time_softpower_in_close	46	~~		

Figure 2: Mapeamento de endereços MC52 V.1.3.0

3 Comunicação por *uart*

A comunicação entre o *pic18f46k40* e o *esp 32 c3* é feita através de porta serie com *baudrate* 9600 (MC50UNI V.0.2.3.X) ou 115200 (MC52 V.1.3.0), *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 [RX] e *GPIO* 19 [TX]).

- *Baudrate*: 9600 ou 115200 (dependendo da central)
- *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

3.1 MC50UNI V.0.2.3.X

```
/* 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);
```

3.2 MC52 V.1.3.0

```
/* Configure parameters of an UART driver,
 * communication pins and install the driver */
uart_config_t uart_config = {
    .baud_rate = 115200,
    .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	
Full	0	read	allows to read from the memory	0x0A	0x00	reading address	retrieved data	0x0D	1
Full	1	write	allows to write into the memory	0x0A	0x01	writing address	written data	0x0D	1
Full	2	programming enable	enables/saves memory changes	0x0A	0x02	0x00	enabled(1)/disabled(0)/blocked(2)	0x0D	1
Full	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 3: 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 3. 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.
Exemplos:

```
(leitura bem sucedida:)
> 0x0A 0x00 0x12 0x00 0x00 0x0D (pedido de leitura do endere o 0x12)
< 0x0A 0x00 0x12 0xAB 0xCD 0x0D (devolu  o da resposta 0xABCD)
< 0x0A 0x03 0x00 0x01 0x00 0x0D (confirma  o de sucesso)

(leitura mal sucedida:)
> 0x0A 0x00 0x14 0x00 0x00 0x0D (pedido de leitura do endere o 0x14)
< 0x0A 0x03 0x00 0x00 0x00 0x0D (confirma  o de insucesso)
```

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 3. No parâmetro *data block* é indicado o novo valor da variável indicada. Deve ser devolvido um comando de confirmação após a execução.
Exemplos:

```
(escrita bem sucedida:)
> 0x0A 0x01 0x12 0x00 0x00 0x0D (pedido de escrita no endereço 0x12)
< 0x0A 0x03 0x00 0x01 0x00 0x0D (confirmação de sucesso)

(escrita mal sucedida:)
> 0x0A 0x01 0x14 0x00 0x00 0x0D (pedido de escrita no endereço 0x14)
< 0x0A 0x03 0x00 0x00 0x00 0x0D (confirmação de insucesso)
```

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.
Exemplos:

```
(Ativar:)
> 0x0A 0x02 0x00 0x00 0x00 0x0D (Toggle ao programming mode)
< 0x0A 0x02 0x00 0x01 0x00 0x0D (Estado ON)
< 0x0A 0x03 0x00 0x01 0x00 0x0D (confirmação de sucesso)

(Desativar:)
> 0x0A 0x02 0x00 0x00 0x00 0x0D (Toggle ao programming mode)
< 0x0A 0x02 0x00 0x00 0x00 0x0D (Estado OFF)
< 0x0A 0x03 0x00 0x01 0x00 0x0D (confirmação de sucesso)

(Bloqueado pelo sistema:)
> 0x0A 0x02 0x00 0x00 0x00 0x0D (Toggle ao programming mode)
< 0x0A 0x02 0x00 0x02 0x00 0x0D (Bloqueado pelo sistema)
< 0x0A 0x03 0x00 0x00 0x00 0x0D (confirmação de insucesso)
```

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).
Exemplos:

```
(Echo da confirmação:)
> 0x0A 0x03 0x00 0x01 0x00 0x0D (envio de confirmação)
< 0x0A 0x03 0x00 0x01 0x00 0x0D (echo da confirmação)
```

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.
Exemplos:

```
(leitura do numero de comandos:)
> 0x0A 0x04 0x00 0x00 0x00 0x0D (Pedido do número de comandos)
< 0x0A 0x04 0x00 0x00 0x03 0x0D (Devolução do número de comandos, ex: 3)
< 0x0A 0x03 0x00 0x01 0x00 0x0D (confirmação de sucesso)
```

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.
Exemplos:

```
(leitura do numero de comandos vazios:)
> 0x0A 0x05 0x00 0x00 0x00 0x0D (Pedido do número de comandos vazios)
< 0x0A 0x05 0x00 0x00 0x02 0x0D (Devolução do número de comandos vazios, ex: 2)
< 0x0A 0x03 0x00 0x01 0x00 0x0D (confirmação de sucesso)
```

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.
Exemplos:

```
(leitura dos comandos:)
> 0x0A 0x06 0x00 0x00 0x00 0x0D (Pedido dos comandos)
< 0x0A 0x06 0x00 0x00 0x00 0x0D (Devolução dos comandos)
< 0x0A 0x06 0x01 0x00 0x03 0x0D (Devolução dos comandos)
< 0x0A 0x06 0x02 0x00 0x04 0x0D (Devolução dos comandos)
< 0x0A 0x03 0x00 0x01 0x00 0x0D (confirmação de sucesso)
```

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.
Exemplos:

```
(leitura dos comandos vazios:)
> 0x0A 0x07 0x00 0x00 0x00 0x0D (Pedido dos comandos vazios)
< 0x0A 0x07 0x00 0x00 0x01 0x0D (Devolução dos comandos vazios)
< 0x0A 0x07 0x01 0x00 0x02 0x0D (Devolução dos comandos vazios)
< 0x0A 0x03 0x00 0x01 0x00 0x0D (confirmação de sucesso)
```

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.
Exemplos:

```
(guardar comando:)
> 0x0A 0x08 0x00 0xAB 0xCD 0x0D (Envio da primeira metade 0xABCD1234)
< 0x0A 0x03 0x00 0x01 0x00 0x0D (confirmação de sucesso)
> 0x0A 0x08 0x00 0x12 0x34 0x0D (Envio da segunda metade 0xABCD1234)
< 0x0A 0x03 0x00 0x01 0x00 0x0D (confirmação de sucesso)

(falha a guardar comando (cheio):)
> 0x0A 0x08 0x01 0xAB 0xCD 0x0D (Envio da primeira metade 0xABCD1234)
< 0x0A 0x03 0x00 0x00 0x00 0x0D (confirmação de insucesso)

(falha a guardar comando (endereço diferente):)
> 0x0A 0x08 0x00 0xAB 0xCD 0x0D (Envio da primeira metade 0xABCD1234)
< 0x0A 0x03 0x00 0x01 0x00 0x0D (confirmação de sucesso)
> 0x0A 0x08 0x02 0x12 0x34 0x0D (Envio da segunda metade 0xABCD1234)
< 0x0A 0x03 0x00 0x00 0x00 0x0D (confirmação de insucesso)
```

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.
Exemplos:

```
(elimina com sucesso:)
> 0x0A 0x09 0x01 0x00 0x00 0x0D (elimina comando na posição 0x01)
< 0x0A 0x03 0x00 0x01 0x00 0x0D (confirmação de sucesso)

(falha a eliminar:)
> 0x0A 0x09 0x02 0x00 0x00 0x0D (elimina comando na posição 0x01)
< 0x0A 0x03 0x00 0x00 0x00 0x0D (confirmação de sucesso)
```

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.
Exemplos:

```
(pede serial com sucesso:)
> 0x0A 0x0A 0x00 0x00 0x00 0x0D (pede serial em 0x00)
< 0x0A 0x0A 0x00 0xAB 0xCD 0x0D (devolução do serial 0xABCD1234)
< 0x0A 0x0A 0x00 0x12 0x34 0x0D (devolução do serial 0xABCD1234)
< 0x0A 0x03 0x00 0x01 0x00 0x0D (confirmação de sucesso)

(falha ao pedir serial:)
> 0x0A 0x0A 0x01 0x00 0x00 0x0D (pede serial em 0x01)
< 0x0A 0x03 0x00 0x00 0x00 0x0D (confirmação de insucesso)
```

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.
Exemplos:

```
(leitura do numero de comandos:)
> 0x0A 0x0B 0x00 0x00 0x00 0x0D (Pedido do número de comandos)
< 0x0A 0x0B 0x00 0x00 0x03 0x0D (Devolução do número de comandos, ex: 3)
< 0x0A 0x03 0x00 0x01 0x00 0x0D (confirmação de sucesso)
```

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.
Exemplos:

```
(leitura do numero de comandos vazios:)
> 0x0A 0x0C 0x00 0x00 0x00 0x0D (Pedido do número de comandos vazios)
< 0x0A 0x0C 0x00 0x00 0x02 0x0D (Devolução do número de comandos vazios, ex: 2)
< 0x0A 0x03 0x00 0x01 0x00 0x0D (confirmação de sucesso)
```

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.
Exemplos:

```
(leitura dos comandos:)
> 0x0A 0x0D 0x00 0x00 0x00 0x0D (Pedido dos comandos)
< 0x0A 0x0D 0x00 0x00 0x00 0x0D (Devolução dos comandos)
< 0x0A 0x0D 0x01 0x00 0x03 0x0D (Devolução dos comandos)
< 0x0A 0x0D 0x02 0x00 0x04 0x0D (Devolução dos comandos)
< 0x0A 0x03 0x00 0x01 0x00 0x0D (confirmação de sucesso)
```

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.
Exemplos:

```
(leitura dos comandos vazios:)
> 0x0A 0x0E 0x00 0x00 0x00 0x0D (Pedido dos comandos vazios)
< 0x0A 0x0E 0x00 0x00 0x01 0x0D (Devolução dos comandos vazios)
< 0x0A 0x0E 0x01 0x00 0x02 0x0D (Devolução dos comandos vazios)
< 0x0A 0x03 0x00 0x01 0x00 0x0D (confirmação de sucesso)
```

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.
Exemplos:

```
(guardar comando:)
> 0x0A 0x0F 0x00 0xAB 0xCD 0x0D (Envio da primeira metade 0xABCD1234)
< 0x0A 0x03 0x00 0x01 0x00 0x0D (confirmação de sucesso)
> 0x0A 0x0F 0x00 0x12 0x34 0x0D (Envio da segunda metade 0xABCD1234)
< 0x0A 0x03 0x00 0x01 0x00 0x0D (confirmação de sucesso)

(falha a guardar comando (cheio):)
> 0x0A 0x0F 0x01 0xAB 0xCD 0x0D (Envio da primeira metade 0xABCD1234)
< 0x0A 0x03 0x00 0x00 0x00 0x0D (confirmação de insucesso)

(falha a guardar comando (endereço diferente):)
> 0x0A 0x0F 0x00 0xAB 0xCD 0x0D (Envio da primeira metade 0xABCD1234)
< 0x0A 0x03 0x00 0x01 0x00 0x0D (confirmação de sucesso)
> 0x0A 0x0F 0x02 0x12 0x34 0x0D (Envio da segunda metade 0xABCD1234)
< 0x0A 0x03 0x00 0x00 0x00 0x0D (confirmação de insucesso)
```

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.
Exemplos:

```
(elimina com sucesso:)
> 0x0A 0x10 0x01 0x00 0x00 0x0D (elimina comando na posição 0x01)
< 0x0A 0x03 0x00 0x01 0x00 0x0D (confirmação de sucesso)

(falha a eliminar:)
> 0x0A 0x10 0x02 0x00 0x00 0x0D (elimina comando na posição 0x01)
< 0x0A 0x03 0x00 0x00 0x00 0x0D (confirmação de sucesso)
```

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.
Exemplos:

```
(pede serial com sucesso:)
> 0x0A 0x11 0x00 0x00 0x00 0x0D (pede serial em 0x00)
< 0x0A 0x11 0x00 0xAB 0xCD 0x0D (devolução do serial 0xABCD1234)
< 0x0A 0x11 0x00 0x12 0x34 0x0D (devolução do serial 0xABCD1234)
< 0x0A 0x03 0x00 0x01 0x00 0x0D (confirmação de sucesso)

(falha ao pedir serial:)
> 0x0A 0x11 0x01 0x00 0x00 0x0D (pede serial em 0x01)
< 0x0A 0x03 0x00 0x00 0x00 0x0D (confirmação de insucesso)
```

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.
Exemplos:

```
(ler todos os parâmetros:)
> 0x0A 0x12 0x00 0x00 0x00 0x0D (pede parâmetros)
< 0x0A 0x00 0x00 0xXX 0xXX 0x0D (devolução de parâmetro em 0x00)
< 0x0A 0x00 0x01 0xXX 0xXX 0x0D (devolução de parâmetro em 0x01)
< 0x0A 0x00 0x02 0xXX 0xXX 0x0D (devolução de parâmetro em 0x02)
...
< 0x0A 0x03 0x00 0x01 0x00 0x0D (confirmação de sucesso)
```

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 número sem ponto decimal
- O tipo integer é um número 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.