Term Project

Documentation

BLETKA

April 16, 2017

Vojtěch Vašek

Contents

1 Glimpse at BLETKA Insides **2**
1.1 Communication with Modules 2
1.1.1 My FLAGS Register 2
1.2 RECMEM 3
1.2.1 Real Time Clock Memory 3
1.2.2 Timestamp Record 4
1.2.3 RECRAM 4
1.2.4 RECROM 5
1.3 CFGMEM 9

2 BLETKA ↔ Android Protocol **9**
2.1 BLETKA Command 9
2.2 BLETKA Commands Preview Table 10
2.2.1 NUM RECS Command 10
2.2.2 LOAD ALL Command 10
2.2.3 LOAD [N] Command 11
2.2.4 REMOVE ALL Command 11
2.2.5 REMOVE [N] Command 11
2.2.6 SAVE [REC] Command 11
2.2.7 SET TIME Command 12
2.2.8 GET TIME Command 12
2.2.9 SET LEDMODE [C] Command 12
2.2.10 SET BEEPMODE [C] Command 12
2.2.11 EXIT Command 12
2.2.12 OK Command 13
2.2.13 ERR Command 13
2.2.14 RECMEM PURGE Command 13
2.2.15 RECMEM PRINT Command 13

Introduction

Bluetooth Low Energy Time Keeper supporting Android shortly just BLETKA is device, which is designed to be able to store accurate current time when requested to do so. This time records are stored inside it and can be fetched later using Bluetooth connection. The device has minimum power requirements, can run on two AAA batteries and can provide its functions through only one single button.

1 Glimpse at BLETKA Insides

Following sections should provide a little bit deeper view into some of the decisions, algorithms and concepts, which BLETKA uses and works with internally.

1.1 Communication with Modules

BLETKA's main part is MCU ATmega328-328P, which is designed as low-power microcontroller. The MCU is connected to Real Time Clock (RTC) module DS1307 (which also contains AT24C32 EEPROM) on TWI/I^{2}C and Bluetooth Low Energy (BLE) module HM-10 on USART. RTC is used to provide accurate time, BLE is used for communication with external devices.

Communication with BLE is interrupt-driven. When a string is about to be send, the procedure always prepares byte into the buffer and goes into sleep mode, from which interrupt will awake it right after the USART buffer is ready to consume another byte.

All functions for I^{2}C and USART communication are written in AVRASM. C language then uses them as a library and provides methods to work with these modules easily.

1.1.1 My FLAGS Register

This register is used to notify about event requirements from buttons. They are set in interrupt routine and should be cleaned when the event is satisfied, because no new event can happen if MF_B*i* or MF_A are not zero. This register (without the MF_T and MF_R bits) is shared with the C code through variable `mint_flags`.

MFLAGS

7	6	5	4	3	2	1	0
MF_A	MF_B4	MF_B3	MF_B2	MF_B1	MF_B0	MF_T	MF_R

MF_A Flag indicating, that device should go to Active state.

MF_B*i* Flag indicating, that record needs to be saved and the event was raised by pushing *i*-th button.

MF_T Flag indicating, that the device is currently transmitting string over USART.

MF_R Flag indicating, that the device is currently receiving string over USART.

1.2 RECMEM

RECMEM is a term derived from RECords MEMory. It consists of two main parts:

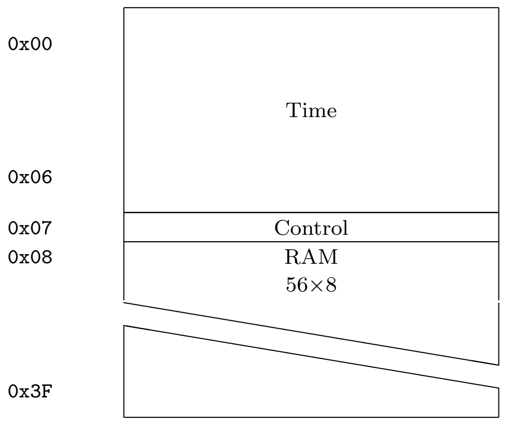
RECRAM Memory not requiring long-term storage, contains helpful metadata about RECMEM, located in SRAM

RECROM Long-term memory, used to store the records, located in EEPROM

More details are presented in following sections.

1.2.1 Real Time Clock Memory

Internal memory of DS1307 module looks like this:



First 7 B are used to store time information in BCD format, Control register is quite unimportant for this application and the rest (RAM part) is left for user to play with. The data remains intact even if the main BLETKA's power source fails under condition, that the backup battery in RTC module still has some juice.

7	6	5	4	3	2	1	0
CH	Seconds 10's			Seconds 1's			
▨	Minutes 10's			Minutes 1's			
▨	$\frac{12}{24}$ bit	Hours 10's		Hours 1's			
▨	▨	▨	▨	▨	Day		
▨	▨	Date 10's		Date 1's			
▨	▨	▨	Month 10's	Month 1's			
Year 10's				Year 1's			

1.2.2 Timestamp Record

In order to use available space more efficiently, following record format was designed:

0 5	6 11	12 16	17 21	22 25	26 31	32 39
SECONDS	MINUTES	HOUR	DAY	MONTH	YEAR	EVOR

Another view, which captures the structure of a record in bytes, would be:

7	6	5	4	3	2	1	0
SECONDS						MINUTES	
MINUTES				HOUR			
HOUR	DAY					MONTH	
MONTH		YEAR					
EVOR							

The record consists of 5 B in total. SECONDS and MINUTES use in BCD format 8 B each. That is totally unnecessary, as we can store the same information using only 6 B. Similarly, using the same idea for the rest of the variables, we can get rid of more bits. On top of that, day in week is an information, which can be derived from the rest of them, therefore *record* is not storing it. With this approach, we can compress the time given by RTC module into 33 bits. To make it exactly 4 B long, the most significant bit of variable YEAR from RTC module is discarded, as 64 years precision looks satisfiable enough.

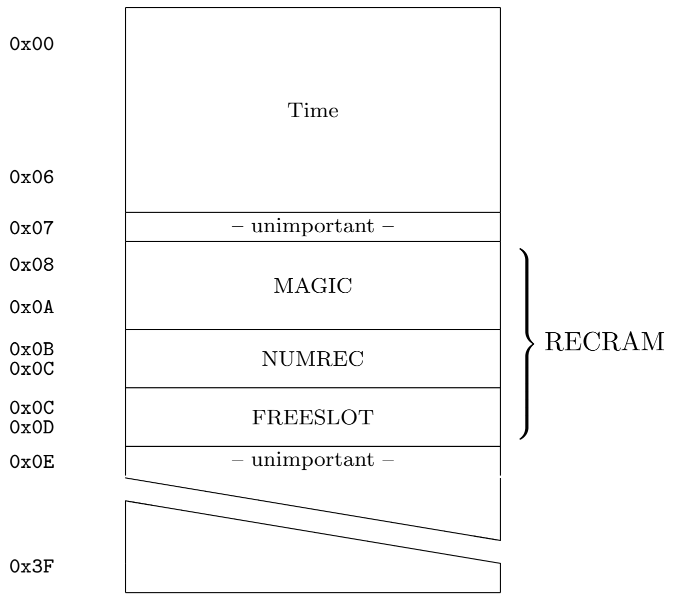
The last byte *EVOR* stays for EVent ORigin and contains unique identifier of what caused the record to be created. It's a copy of MFLAGS register (described in 1.1.1) without the last two bits. If only single button is wired, this byte is unnecessary and the feature to use it should be disabled to increase the number of records, which can be stored in memory in total.

1.2.3 RECRAM

The RECRAM is present in the RTC DS1307's RAM space. As it was already mentioned above, this has an advantage, that the data stays valid as long as the backup battery in RTC module provides enough power. That is always longer than the main BLETKA's power source, as the backup battery charges itself from the main power source. Also the RTC module consumes less power than the main circuit and the RECROM is present on the same module as well (therefore RECMEM can be located somehow together).

On the other hand, the system is designed so that this information can be stored elsewhere with no problems.

Our new view on the DS1307 memory can now be modified to:



Here MAGIC is a string of 3 characters:

MAGIC

B	T	K
0	1	2

It's used to determine, if the RECRAM is initialized and if the data is corrupted or not. If not present, BLETKA will suppose, that metadata are corrupted, it will recalculate the information and write the calculated metadata with MAGIC back to RECRAM.

The two main metadata variables are unsigned integer values consisting of 2 B each:

NUMREC		FREESLOT	
NR_H	NR_L	FS_H	FS_L
0	1	0	1

the purpose of these will be expained in the following section 1.2.4.

1.2.4 RECROM

In order to store records, AT24C32 I^{2}C Serial EEPROM was chosen to be used. The main reasons for this decision instead of using Atmega328's internal EEPROM are:

- this memory is available on the same module where RTC is located, therefore it's already connected to the I^{2}C line
- it provides 4 kB, whereas Atmega328 provides 1 kB

- it has endurance at least 10^6 write cycles, whereas Atmega328 states 10^5

If 5 B type of record is used, this means, that 819 records can stay in the RECROM.

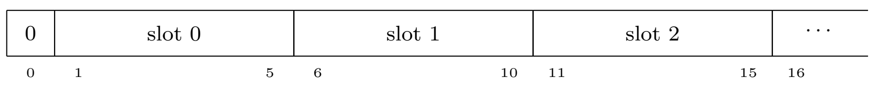
RECROM is again divided into sections. The first part is called HEADER, it fits into the first 1 B, the rest of the RECROM is either *slot* or unused space.

Slot is simply a block of length `sizeof(record)` and it is designed to store a whole single record. Maximum number of slots (and therefore records) in RECROM is given by, and in this case equals

$$\frac{RECROM\_BYTES - RECROM\_HEADER\_LENGTH}{RECORD\_LENGTH} = \frac{4096-1}{5} = 819.$$

Variable HEADER contains number from range $[0; sizeof(record)-1]$. This number defines, where the first slot is located, it specifies it by shift to higher address relative to the end of HEADER. For the rest of the slots, the next slot is always located right after the previous one. HEADER is initialized to value 0, therefore the first slot initially starts at address `0x01`.

Initial RECROM setup:



RECRAM contains two integeres, which describe basic information about the RECROM and can be used to speed up the process of saving a new record. The NUMREC contains number of records currently present in the RECROM, variable FREESLOT contains number of a slot, which is currently waiting to be used for storing next record. Both of them are therefore initially 0.

Storing New Record When a new record should be stored to RECMEM, metadata from RECRAM are loaded and FREESLOT along with HEADER is used to determine position of the free slot address in RECROM. Record is stored to this slot and both NUMREC and FREESLOT are incremented by one.

In case that RECROM is full, BLETKA will notify about this situation by making an error sound.

Removing a Record To make minimum number of writes, the deletion is done simply by writing `0x00` into fourth byte of the slot containing the record, which should be deleted. NUMREC variable is decreased, FREESLOT stays unchanged, meaning no new record can be stored in this slot.

RECMEM Rotation When FREESLOT reaches slot number which is out of range, RECMEM must be so-called *rotated.* This operation completely destroys all records, if needed, they should therefore be transmitted before doing this operation to external device.

Rotation updates the information in HEADER, it updates this variable to $HEADER = (HEADER + 1)$ % `sizeof(record)`, meaning the new slots will be shifted to a new position. After this, it's made sure, that every third or fourth byte of every slot is containing `0x00`, if not `0x00` is written to the third bit.

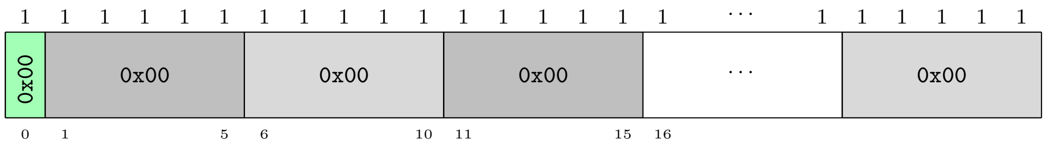
This whole procedure was designed to make as equal number of writes to each EEPROM byte cell as possible. Following diagram shows, that this algorithm satisfies this condition exept few bytes on the fringe in the whole memory, these few bytes actualy receive even less number of writes, therefore they are not the bottleneck for EEPROM failure. As a result of this algorithm, it's expected, that the memory can achieve storing at least $819 \cdot 5 \cdot 10^6$ records in total instead of 10^6 if naive deletion of all bytes would be used. (Meaning we can store at least $(819 \cdot 5 \cdot 10^6)/((2063 - 2017 + 1) \cdot 365) = 238705$ records every day until the device (RTC interpretation) will not work as opposed to only 58 if naive implementation would be chosen.)

The procedure can take quite a long time, for selected 4 kB RECROM, it should take about $(20 \cdot (4096/5))/1000 = 16.384\,\mathrm{s}$. (20 ms is delay time required by the EEPROM after each sequential write.) Better approach would made use of page writes, with possible theoretical speed up (not including page read operations) to $(20 \cdot (4096/5/32))/1000 = 0.512\,\mathrm{s}$

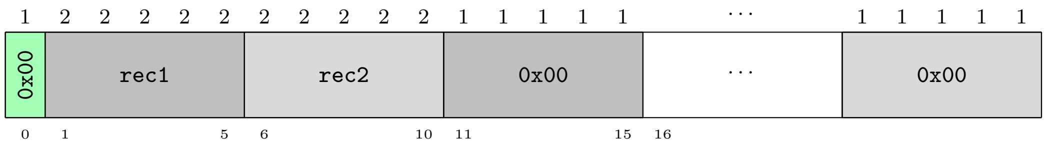
Recovery from Corruption When the battery level in RTC module goes under aproximately 1.8 V or RECMEM is corrupted any other way, the metadata are not available and must be recalculated again when the power is back on. This procedure is very much possible with as little effort as going once through the whole RECROM (nevertheless, it's too demanding to do it this way everytime we need to store a new record). The procedure just looks at third and fourth byte in every slot using HEADER shift information.

In the following diagrams, the RECROM is drawn linearly, first row of numbers shows number of write operations into the byte cell under the number, green color is used to highlight the HEADER, shades of gray show current slot positions, red color highlights currently unused bytes.

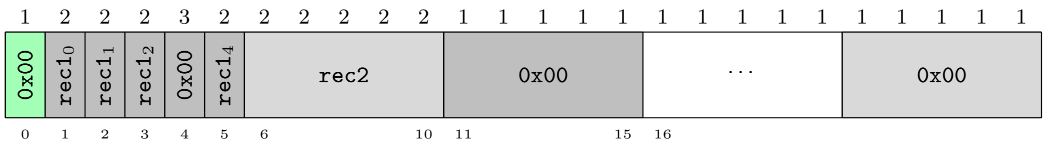
`recmem_purge();` – writes `0x00` to each cell



`save_rec(rec1); save_rec(rec2);` – puts two new records into the memory

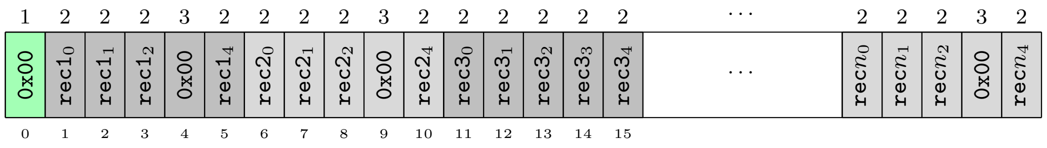


`destroy_rec(0);` – deletes record at given slot position

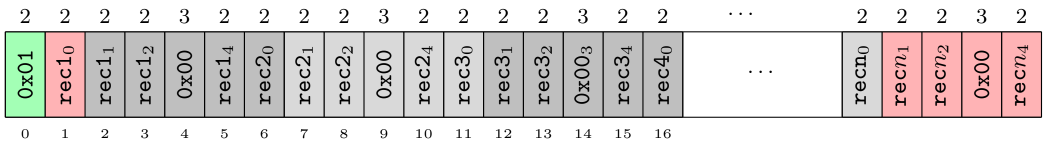


After filling the whole RECROM with records, some of them might be destroyed, let's see how rotation operation handles the situation:

Full RECROM:



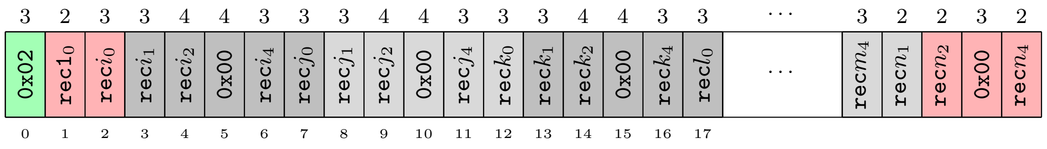
`recmem_rotate();` – performs the rotation operation



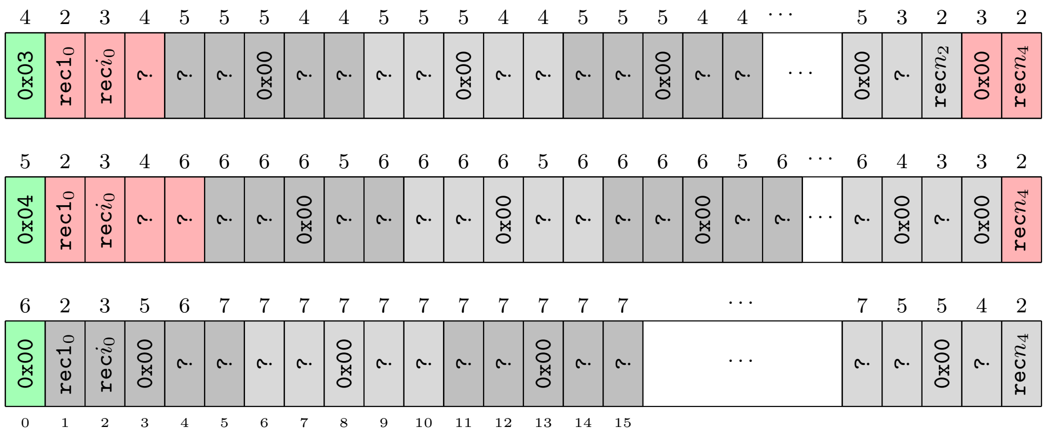
As rotation deletes only when the slot is containing valid record and the write is made into the third byte of a slot after HEADER shift, each slot will have the same number of writes into its cells.

When the RECROM is full once again and rotation is performed:

`recmem_rotate();`



After doing this 2 more times, let's look at the more interesting fifth rotation:



As we can see, majority of the bytes receives the same number of writes in their byte cells. One could even use the unused bytes to store another record to make the most out of this. The third and fourth bytes of the slot were used intentionaly, because their valid values can never contain `0x00` , as third byte contains day in month (1–31) and fourth contains year (17–63), which will not be 0 until April 2064.

1.3 CFGMEM

This term stays for ConFiGuration MEMory and it's located in device's internal EEPROM. It contains following configuration:

LED How, if, what and when the LED should indicate actions performed by BLETKA.

SPEAKER How, if, what and when the speaker should indicate actions performed by BLETKA.

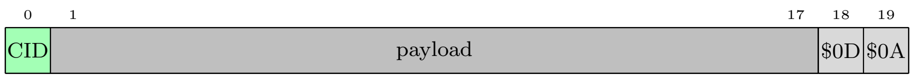
These features are currently not implemented at all.

2 BLETKA ↔ Android Protocol

When a button is pushed for a longer time (roughly 0.5 s), BLETKA will wake up its BLE module and prepare itself for communication with external device. This state of BLETKA is in this document called Active. Set of commands, which BLETKA understands is presented in following sections. Simple Android application was written to demostrate functionality of this process of communication in Active mode.

2.1 BLETKA Command

One single command consists of frames containing at most 20 B, but at least 3 B. Each command starts with unique ID named CID (Command Identifier) and ends with characters \r\n.



This format was selected based on the fact, that BLE module HM-10 internally works with 20 B long batches and its own commands must end with \r \n = `0x0D 0x0A` characters. The `0x0A` character is used to recognize end of the command and therefore cannot be used elsewhere in its body. All the commands are designed so that `0x0A` does not ever have to be part of the payload.

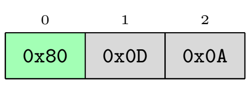
2.2 BLETKA Commands Preview Table

CID hex	CID dec	Command Name	Command Description
0x80	128	NUM RECS	Returns RECMEM metadata
0x81	129	LOAD ALL	Returns all records
0x82	130	LOAD [N]	Returns specified record
0x83	131	REMOVE ALL	Removes all records from RECMEM and rotates
0x84	132	REMOVE [N]	Removes selected record
0x85	133	SAVE [REC]	Saves record to memory
0x86	134	SET TIME	Sets internal time of BLETKA
0x87	135	GET TIME	Gets internal time of BLETKA
0x88	136	SET LEDMODE [C]	Sets LED configuration
0x89	137	SET BEEPMODE [C]	Sets speaker configuration
0x8A	138	EXIT	Exits the Active mode
0x8B	139	OK	Confirmation message
0x8C	140	ERR	Error message
0x8D	141	RECMEM PURGE	Writes delete character to whole RECMEM
0x8E	142	RECMEM PRINT	Prints content of RECMEM

2.2.1 NUM RECS Command

Returns RECMEM metadata.

The first message returned by BLETKA will contain the content of RECRAM. Second message will contain number of records in RECMEM and freeslot variable in human-readable form.



2.2.2 LOAD ALL Command

Returns all records.

This command prompts for all slots in RECMEM. BLETKA will return all slots each in single message of 5 B in the same order as they are stored.

Some of these records can be invalid, invalid records have `0x00` byte either at third or fourth position.

0	1	2
`0x81`	`0x0D`	`0x0A`

2.2.3 LOAD [N] Command

Returns specified record.

This command can be used to retrieve single slot, specified by its position. The position should be from range $[0; NUMREC - 1]$, to encode the unsigned integer number in command, use human-readable decimal form. The last numeral must be followed by \r \n characters.

0	1	2	3				
`0x82`	N_n	N_{n-1}	N_{n-2}	$\cdots$	N_0	`0x0D`	`0x0A`

The value is then calculated by $N = \sum_{i=0}^{n} (N_i - \text{'0'}) \cdot 10^i$.

2.2.4 REMOVE ALL Command

Removes all records from RECMEM and rotates.

This command will make sure, that every record is destroyed, the RECMEM is rotated to prepare place for a new set of records. This operation should be used after all records are saved externally and the RECMEM is full.

0	1	2
`0x83`	`0x0D`	`0x0A`

2.2.5 REMOVE [N] Command

Removes selected record.

Destroys the record by writing `0x00` character to the fourth byte of a slot at given position. The position form is the same as in LOAD [N] Command2.2.3.

0	1	2	3				
`0x84`	N_n	N_{n-1}	N_{n-2}	$\cdots$	N_0	`0x0D`	`0x0A`

2.2.6 SAVE [REC] Command

Saves record to memory.

Given uncompressed record included in payload section of the command will be put into the RECMEM if free memory is still available. The first 7 bytes of payload should be in BCD form as presented in section 1.2.1.

It should be noted, that EVOR must not be equal to `0x0A` as that would cause problems as described in 2.1.

0	1	2	3	4	5	6	7	8	9	10
`0x85`	SEC	MIN	HOUR	DAY	DATE	MON	YEAR	EVOR	`0x0D`	`0x0A`

2.2.7 SET TIME Command

Sets internal time of BLETKA.

Stores given payload in BCD form into the RTC module. Content of the payload must be exactly what is going to be stored into the RTC module Time registers. The details about each bit can be found in previous section about the RTC module1.2.1.

0	1	2	3	4	5	6	7	8	9
`0x86`	rtc$_0$	rtc$_1$	rtc$_2$	rtc$_3$	rtc$_4$	rtc$_5$	rtc$_6$	`0x0D`	`0x0A`

2.2.8 GET TIME Command

Gets internal time of BLETKA.

Reads content of the RTC module Time registers (as described in 1.2.1), makes a compressed version (described in section 1.2.2) and sends it in a message back to the connected device.

0	1	2
`0x87`	`0x0D`	`0x0A`

2.2.9 SET LEDMODE [C] Command

Sets LED configuration.

Currently not implemented.

0	1	2	3
`0x88`	`?`	`0x0D`	`0x0A`

2.2.10 SET BEEPMODE [C] Command

Sets speaker configuration.

Currently not implemented.

0	1	2	3
`0x89`	`?`	`0x0D`	`0x0A`

2.2.11 EXIT Command

Exits the Active mode.

Tells BLETKA, that the device will disconnect shortly after this message. After sending this command, no other commands will be performed until new connection to Active mode is established.

0	1	2
`0x8A`	`0x0D`	`0x0A`

2.2.12 OK Command

Confirmation message.

Currently not used.

0	1	2
`0x8B`	`0x0D`	`0x0A`

2.2.13 ERR Command

Error message.

Currently not used.

0	1	2
`0x8C`	`0x0D`	`0x0A`

2.2.14 RECMEM PURGE Command

Writes delete character to whole RECMEM.

This command will perform write operation of `0x00` character into the whole RECROM. RECRAM is set to the default values as well.

0	1	2
`0x8D`	`0x0D`	`0x0A`

2.2.15 RECMEM PRINT Command

Prints content of RECMEM.

After sending this command, BLETKA will return back first $5 \cdot 20\,\mathrm{B}$ of RECROM. It can be used mainly for debug purposes.

0	1	2
`0x8E`	`0x0D`	`0x0A`