EBmonitor Target Integration

by Gerard Zagema (EmBitz okt 2025)

This document explains how to use `ebmon.c` in your embedded target, configure its defines, and control **blocking vs non-blocking mode**.

---

1. Adding EBmonitor to Your Target

1. Copy `ebmon.c` and `ebmon.h` into your target project.
2. Include the header in your source files:

```
#include "ebmon.h"
```

3. Link `ebmon.c` in your build system so it compiles with your firmware.

---

2. Configurable Defines

Several macros control buffer sizes, initialization, and behavior:

Define	Purpose	Default
`STDOUT_BUFFERSIZE`	Size of stdout buffer	256
`STDIN_BUFFERSIZE`	Size of stdin buffer	16
`NO_EBMON_INIT`	Disable automatic initialization	Not defined
`EBMON_WRITE_WAIT`	Enable blocking writes if buffer is full	Not defined

Example Configuration

```
#define STDOUT_BUFFERSIZE 1024
#define STDIN_BUFFERSIZE  512
#define EBMON_WRITE_WAIT  // enable blocking mode for writes
```

Place these defines in your project configuration header or at the top of `ebmon.c` before includes.

---

3. Initialization

EBmonitor automatically initializes buffers on the first `_write()` call unless `NO_EBMON_INIT` is defined.

You can manually initialize using:

```
#ifndef NO_EBMON_INIT
ebmonitor_init();
#endif
```

This sets up `_eb_monitor_stdout` and clears the screen on the host by sending a formfeed `\f`.

---

4. Using STDIO Pipes

EBmonitor exposes two primary pipes:

- `_eb_monitor_stdout` — write data to host
- `_eb_monitor_stdin` — read data from host

Writing to stdout

```
_write(1, "Hello World\n", 12);
```

- `file` argument is ignored, `ptr` points to buffer, `len` is number of bytes.
- Sending `\f` to `_write()` clears the host screen.

Reading from stdin

```
char buf[32];
int n = _read(0, buf, sizeof(buf));
if(n > 0) {
    // process input
}
```

- Returns `-1` if no data is available.

Flushing a pipe

```
EBmonitor_flush(stdout);
EBmonitor_flush(stdin);
```

- Clears the respective buffer.

Check if data is available

```
if(EBmonitor_kbhit()) {
    // data ready
}
```

---

5. Blocking vs Non-Blocking Mode

EBmonitor can operate in **blocking** or **non-blocking** mode depending on `EBMON_WRITE_WAIT`:

Mode	Behavior
Blocking (`EBMON_WRITE_WAIT` defined)	`_write()` waits if buffer is full until space is available.
Non-Blocking (default)	`_write()` returns the number of characters actually written, may drop characters if buffer is full.

Choosing the Mode

- **Blocking Mode:** Use when data integrity is critical and you cannot afford to drop characters.
- **Non-Blocking Mode:** Use when your firmware must continue running without delay even if the buffer is temporarily full.

---

6. Notes

- Buffers are aligned to 4 bytes for performance on most MCUs.
- `_eb_monitor_stdout` and `_eb_monitor_stdin` are circular buffers, so always respect the `head` and `tail` indices if you directly access them.
- To clear the host terminal screen, send a formfeed character `\f` via `_write()`.

---

7. Example Firmware Loop

```
char input[64];
printf("\f"); // clear terminal screen
while(1) {
    if(EBmonitor_kbhit()) {
        int n = _read(0, input, sizeof(input));
        // process input
    }

    _write(1, "Heartbeat\n", 10);
    delay_ms(1000);
}
```

This simple loop demonstrates reading commands from the host and writing periodic status updates.