# BOS Bitz Operating System

## Commands *use with any serial terminal software*

`command parameter1 parameter2 …`

**ping**

Ping a module.

**update**

Put the module in bootloader mode to update its firmware.

**reset**

Reset (reboot) the module.

**help**

Display help information for Commands available on this module. Type `help params` to display available parameters and their possible values.

**name** *alias*

Name the module with an *alias* name. Max 10 characters.

**status**

Display current module status.

**info**

Display current array information.

**explore**

Explore the array and create a topology file.

**add-button** *type port*

Define a button/switch with a specific *type* at this *port*. Button type can be `momentary-no` (momentary switch, naturally-open), `momentary-nc` (momentary switch, naturally-closed), `onoff-no` (on/off switch, naturally-open) and `onoff-no` (on/off switch, naturally-closed).

**removebutton** *port*

Remove a button that was previously defined at this *port*. This restores the port into its default state (array port).

**task-stats**

Display a table with information on each task in the module.

**run-time-stats**

Display a table with information on how much time each task has taken so far.

## Commands *continued*

**scast** *srcPort srcModule dstPort dstModule direction count timeout*

Start a single-cast (single-input-single-output) DMA stream. Stream starts from *srcPort* (source port) in *srcModule* (source module), to *dstPort* (destination port) in dstModule (destination module) with stream *direction* going (`forward`, `backward`, or `bidirectional`). Stream is disabled after *count* ($1-2^{32}$) bytes is transferred or a *timeout* ($1-2^{32}$) ms, whichever comes first.

**if** *bx.event*
*command1*
*command2*
…
**end if**

Execute a set of *commands* on one of the button *bx* events: `clicked`, `double clicked`, `pressed for X`, `released for Y`, where `X` and `Y` are in seconds (`1-254`). You can define maximum of three *pressed* events and three *released* events.

**get** *parameterName*

Display the current value of parameter `parameterName`.

**set** *parameterName value*

Assign the parameter `parameterName` a new `value`. The new value is saved to to the emulated EEPROM.

**default params**

Set all parameters to their default value.

### *Examples*

```
name alpha

scast p1 alpha p5 beta forward 100 1000

scast p4 #001 p4 #002 bidirectional 1000 3600000

addbutton momentary-no p3

if b2.clicked
ping
end if

if b5.pressed for 3
color red 30
end if

set bos.response none
```

# Bitz Operating System BOS

## Messages *for inter-array communication*

`code, parameter1 [value], parameter2 [value], …`

`CODE_unknown_message`

`CODE_ping`

`CODE_ping_response`
Ping textual response.

`CODE_IND_toggle`
Toggle indicator LED.

`CODE_hi, myID, myPort, myPN`
Identify a module. It asks about module ID, connection port and part number (PN).

`CODE_hi_response, myID, myPort, myPN`

`CODE_explore_adj`
Explore adjacent modules.

`CODE_explore_adj_response, nonVacantPort, neighborID, neighborPort, neighborPN, …`
Information about adjacent modules.

`CODE_port_dir, P1direction [NORMAL, REVERSED], P2direction, P3direction, …`
Change module port directions.

`CODE_module_id, targetModule [0:me, 1:neighbor], newID, neighborPort`
Change the ID of a module or one of its neighbors.

`CODE_topology, arrayTopology`
Share and save this array topology to a volatile memory.

`CODE_read_port_dir`
Ask for module ports direction.

`CODE_read_port_dir_response, reversedPort1, reversedPort2, …`
Reply with reversed ports.

`CODE_exp_eeprom`
Store array exploration results in a non-volatile memory.

`CODE_CLI_command, commandString`
Forward a CLI Command to another module.

`CODE_CLI_response, responseString`
CLI Response textual string.

`CODE_DMA_channel, count 4th byte (MSB), count 3rd byte, count 2nd byte, count 1st byte, timeout 4th byte (MSB), timeout 3rd byte, timeout 2nd byte, timeout 1st byte, srcPort1|dstPort1, (optional) srcPort2|dstPort2, (optional) srcPort3|dstPort3`
Activate local port-to-port DMA channels (single-input-single-output or single-input-multi-output). Note that modules will power-cycle after receiving this command.

## Messages *continued*

`CODE_DMA_scast_stream, count 4th byte (MSB), count 3rd byte, count 2nd byte, count 1st byte, timeout 4th byte (MSB), timeout 3rd byte, timeout 2nd byte, timeout 1st byte, direction, srcPort, dstModule, dstPort`
Start a single-cast (single-input-single-output) DMA stream across the array (starting from this module).

### *Examples*

```
messageParams[0] = (uint8_t)(1000>>24);
messageParams[1] = (uint8_t)(1000>>16);
messageParams[2] = (uint8_t)(1000>>8);
messageParams[3] = (uint8_t)(1000);
messageParams[4] = (uint8_t)(600000>>24);
messageParams[5] = (uint8_t)(600000>>16);
messageParams[6] = (uint8_t)(600000>>8);
messageParams[7] = (uint8_t)(600000);
messageParams[8] = FORWARD;
messageParams[9] = (P1<<4)|P3;
messageParams[9] = (P1<<4)|P5;
SendMessageToModule(2, CODE_DMA_channel, 7);
SendMessageToModule(2, CODE_H01R0_OFF, 0);
```

## APIs *getting your hands dirty!*

`output API_function(inputs)`

**`IND_toggle`**`()`
**`IND_ON`**`()`
**`IND_OFF`**`()`
Toggle/on/off indicator LED (macros).

**`IND_blink`**`(t)`
Blink indicator LED for `t` msec (macro). Use after starting the scheduler.

**`RTOS_IND_blink`**`(t)`
Blink indicator LED for `t` msec (RTOS-safe macro). Use after starting the scheduler.

**`NumberOfHops`**`(i)`
Get number of hops to module `i` in the array (macro). The `FindRoute` API must be called first.

**`Delay_us`**`(t)`
RTOS-safe (1-$2^{16}$) µsec delay (macro). Use before and after starting the scheduler.

**`Delay_ms_no_rtos`**`(t)`
RTOS-safe (1-$2^{16}$) µsec delay (macro). Use before and after starting the scheduler.

**`Delay_ms`**`(t)`
Non-RTOS-safe (1-$2^{32}$) msec delay (macro). Use after starting the scheduler.

# BOS Bitz Operating System

## APIs *continued*

**Delay_s**(t)
Non-RTOS-safe (1-$2^{29}$) sec delay (macro). Use after starting the scheduler.

*UART_HandleTypeDef** **GetUart**(uint8_t port)
Get the UART of a given array port.

*uint8_t* **GetPort**(UART_HandleTypeDef *huart)
Get the array port of a given UART.

*BOS_Status* **SendMessageToModule**(uint8_t dst, uint16_t code, uint8_t numberOfParams)

*BOS_Status* **SendMessageFromPort**(uint8_t port, uint8_t src, uint8_t dst, uint16_t code, uint8_t numberOfParams)

**StartMicroDelay**(uint16_t Delay)
Load and start micro-second delay counter.

*BOS_Status* **Explore**()

*BOS_Status* **ExploreNeighbors**()

**SwapUartPins**(UART_HandleTypeDef *huart, uint8_t direction)
Swap UART pins. direction = NORMAL (TX top) or REVERSED (RX top).

*uint8_t* **FindRoute**(uint8_t sourceID, uint8_t desID)
Find the shortest route to a module using Dijkstra's algorithm. Returns the port leading to that route.

**DisplayTopology**(uint8_t port)
Display array topology in human-readable format via the CLI.

**DisplayPortsDir**(uint8_t port)
Display ports direction in human-readable format via the CLI.

**DisplayModuleStatus**(uint8_t port)
Display current module status report via the CLI.

*uint8_t* **GetID**(char* string)
Extract module ID from its alias, ID string or keyword.

*BOS_Status* **NameModule**(uint8_t module, char* alias)
Name a module with an alias.

*BOS_Status* **ReadPortsDir**()
Read ports direction when a pre-defined topology file is used.

## APIs *continued*

*BOS_Status* **UpdateMyPortsDir**()
Update module ports direction based on stored in EEPROM.

*BOS_Status* **StartScastDMAStream**(uint8_t srcM, uint8_t dstM, uint8_t srcP, uint8_t dstP, , uint8_t direction, uint32_t count, uint32_t timeout)
Start a single-cast DMA stream across the array.

*BOS_Status* **AddPortButton**(uint8_t buttonType, uint8_t port)
Define a new button attached to one of array ports. buttonType = MOMENTARY_NO, MOMENTARY_NC, ONOFF_NO, ONOFF_NC.

*BOS_Status* **RemovePortButton**(uint8_t port)
Undefine a button attached to one of array ports and restore the port to default state.

*BOS_Status* **SetButtonEvents**(uint8_t port, uint8_t clicked, uint8_t dbl_clicked, uint8_t pressed_x1sec, uint8_t pressed_x2sec, uint8_t pressed_x3sec, uint8_t released_y1sec, uint8_t released_y2sec, uint8_t released_y3sec)
Setup button events and callbacks.

### *Examples*

```
IND_toggle();

Delay_s(3);

Port = FindRoute(srcM, dstM);

NameModule(1, “alpha”);

id = GetID(“alpha”);

UART_HandleTypeDef* uart = GetUart(P2);

AddPortButton(MOMENTARY_NO, P1);

SetButtonEvents(P1, 1, 1, 1, 3, 0, 10, 0, 0);
```