IC functions listed alphabetically:

	RCX	allbrake: <void> ()	Motors
HB	RCX	alloff: <void> ()	Motors
HB		analog: <int> (<int>)	Sensors
HB	RCX	ao: <void> ()	Motors
HB	RCX	atan: float (<float>)	Math
	RCX	battery_volts: <float> ()	Power
HB	RCX	beep: <void> ()	Sound
HB	RCX	beeper_off: <void> ()	Sound
HB	RCX	beeper_on: <void> ()	Sound
HB	RCX	bk: <void> (<int>)	Motors
	RCX	brake: <void> (<int>)	Motors
HB		clear_digital_out: <int> (<int>)	DIO
HB	RCX	cos: float (<float>)	Math
HB	RCX	defer: <void> ()	Processes
HB	RCX	digital: <int> (<int>)	Sensors
HB		disable_encoder: <void> (<int>)	Sensors
HB		enable_encoder: <void> (<int>)	Sensors
HB		exp: float (<float>)	Math
HB		exp10: float (<float>)	Math
HB	RCX	fd: <void> (<int>)	Motors
	RCX	getticksremaining: <int> ()	Time
HB	RCX	hog_processor: <void> ()	Processes
HB		init_expbd_servos: <int> (<int>)	Motors
HB	RCX	kill_process: void (<int>)	Processes
HB		knob: <int> ()	Sensors
	RCX	light: <int> (<int>)	Sensors
	RCX	light_passive: <int> (<int>)	Sensors
HB		log: float (<float>)	Math
HB		log10: float (<float>)	Math
	RCX	moreticks: <void> (<int>)	Processes
HB	RCX	motor: <void> (<int>, <int>)	Motors
HB	RCX	mseconds: long ()	Time
HB	RCX	msleep: <void> (<long>)	Time
HB	RCX	off: <void> (<int>)	Motors
	RCX	poweroff: <void> ()	Power
	RCX	prgm_button: <int> ()	Sensors
HB	RCX	printf: void (<char[]>, ...)	Output
HB		random: <int> (<int>)	Math
HB		read_encoder: <int> (<int>)	Sensors
	RCX	reset: <void> ()	Power
HB		reset_encoder: <void> (<int>)	Sensors
HB	RCX	reset_system_time: <void> ()	Time
HB		seconds: <float> ()	Time
HB	RCX	set_beeper_pitch: <void> (<float>)	Sound
HB		set_digital_out: <int> (<int>)	DIO
	RCX	setticks: <void> (<int>)	Time
HB	RCX	sin: float (<float>)	Math
HB	RCX	sleep: <void> (<float>)	Time
HB		sonar: <int> ()	Sensors
HB	RCX	sqrt: float (<float>)	Math
HB	RCX	start_button: <int> ()	Sensors
HB		start_press: <void> ()	Sensors
HB	RCX	stop_button: <int> ()	Sensors
HB		stop_press: <void> ()	Sensors
HB	RCX	tan: float (<float>)	Math
HB		test_digital_out: <int> (<int>)	DIO
HB	RCX	tone: <void> (<float>, <float>)	Sound
	RCX	touch: <int> (<int>)	Sensors
	RCX	view_button: <int> ()	Sensors

IC functions listed by category:

HB		clear_digital_out: <int> (<int>)	DIO
HB		set_digital_out: <int> (<int>)	DIO
HB		test_digital_out: <int> (<int>)	DIO
HB	RCX	atan: float (<float>)	Math
HB	RCX	cos: float (<float>)	Math
HB		exp: float (<float>)	Math
HB		exp10: float (<float>)	Math
HB		log: float (<float>)	Math
HB		log10: float (<float>)	Math
HB		random: <int> (<int>)	Math
HB	RCX	sin: float (<float>)	Math
HB	RCX	sqrt: float (<float>)	Math
HB	RCX	tan: float (<float>)	Math
	RCX	allbrake: <void> ()	Motors
HB	RCX	alloff: <void> ()	Motors
HB	RCX	ao: <void> ()	Motors
HB	RCX	bk: <void> (<int>)	Motors
	RCX	brake: <void> (<int>)	Motors
HB	RCX	fd: <void> (<int>)	Motors
HB		init_expbd_servos: <int> (<int>)	Motors
HB	RCX	motor: <void> (<int>, <int>)	Motors
HB	RCX	off: <void> (<int>)	Motors
HB	RCX	printf: void (<char[]>, ...)	Output
	RCX	battery_volts: <float> ()	Power
	RCX	poweroff: <void> ()	Power
	RCX	reset: <void> ()	Power
HB	RCX	defer: <void> ()	Processes
HB	RCX	hog_processor: <void> ()	Processes
HB	RCX	kill_process: void (<int>)	Processes
	RCX	moreticks: <void> (<int>)	Processes
HB		analog: <int> (<int>)	Sensors
HB	RCX	digital: <int> (<int>)	Sensors
HB		disable_encoder: <void> (<int>)	Sensors
HB		enable_encoder: <void> (<int>)	Sensors
HB		knob: <int> ()	Sensors
	RCX	light: <int> (<int>)	Sensors
	RCX	light_passive: <int> (<int>)	Sensors
	RCX	prgm_button: <int> ()	Sensors
HB		read_encoder: <int> (<int>)	Sensors
HB		reset_encoder: <void> (<int>)	Sensors
HB		sonar: <int> ()	Sensors
HB	RCX	start_button: <int> ()	Sensors
HB		start_press: <void> ()	Sensors
HB	RCX	stop_button: <int> ()	Sensors
HB		stop_press: <void> ()	Sensors
	RCX	touch: <int> (<int>)	Sensors
	RCX	view_button: <int> ()	Sensors
HB	RCX	beep: <void> ()	Sound
HB	RCX	beeper_off: <void> ()	Sound
HB	RCX	beeper_on: <void> ()	Sound
HB	RCX	set_beeper_pitch: <void> (<float>)	Sound
HB	RCX	tone: <void> (<float>, <float>)	Sound
	RCX	getticksremaining: <int> ()	Time
HB	RCX	mseconds: long ()	Time
HB	RCX	msleep: <void> (<long>)	Time
HB	RCX	reset_system_time: <void> ()	Time
HB		seconds: <float> ()	Time
	RCX	setticks: <void> (<int>)	Time
HB	RCX	sleep: <void> (<float>)	Time

Library Function Descriptions (alphabetic order):

(gold is for RCX specific, orange for HB specific, dark blue for IC functions common to both)

RCX `allbrake: <void> ()`

HB RCX `alloff: <void> ()`

`void alloff()`

Turns off all motors. `ao` is a short form for `alloff`.

HB `analog: <int> (<int>)`

`int analog(int p)`

Returns value of sensor port numbered **`p`**. Result is integer between 0 and 255. If the `analog()` function is applied to a port that is implemented digitally in hardware, then the value 255 is returned if the *hardware* digital reading is 1 (as if a digital switch is open, and the pull up resistors are causing a high reading), and the value 0 is returned if the *hardware* digital reading is 0 (as if a digital switch is closed and pulling the reading near ground). Ports are numbered as marked. Note that ports 16-22 are floating, so without a sensor inserted, the value cannot be predicted.

HB RCX `ao: <void> ()`

`void ao()`

Turns off all motors.

HB RCX `atan: float (<float>)`

`float atan(float angle)`

Returns arc tangent of angle. Angle is specified in radians; result is in radians.

RCX `battery_volts: <float> ()`

HB RCX `beep: <void> ()`

`void beep()`

Produces a tone of 500 Hertz for a period of 0.3 seconds. Returns when the tone is finished.

HB RCX `beeper_off: <void> ()`

`void beeper_off()`

Turns off the beeper.

HB RCX `beeper_on: <void> ()`

`void beeper_on()`

Turns on the beeper at last frequency selected by the former function. The beeper remains on until the `beeper_off` function is executed.

HB RCX `bk: <void> (<int>)`

`void bk(int m)`

Turns motor m on in the backward direction. Example: `bk(1);`

RCX `brake: <void> (<int>)`

HB `clear_digital_out: <int> (<int>)`

HB RCX `cos: float (<float>)`

`float cos(float angle)`

Returns cosine of **`angle`**. Angle is specified in radians; result is in radians.

HB RCX `defer: <void> ()`

`void defer()`

Makes a process swap out immediately after the function is called. Useful if a process knows that it will not need to do any work until the next time around the scheduler loop. `defer()` is implemented as a C built-in function.

HB RCX `digital: <int> (<int>)`

`int digital(int p)`

Returns the value of the sensor in sensor port p, as a true/false value (1 for true and 0 for false). Sensors are expected to be *active low*, meaning that they are valued at zero volts in the active, or true, state. Thus the library function returns the inverse of the actual reading from the digital hardware: if the reading is zero volts or logic zero, the digital() function will return true.

HB `disable_encoder: <void> (<int>)`

`void disable_encoder(int encoder)`

Disables the given encoder and prevents it from counting. Each shaft encoder uses processing time every time it receives a pulse while enabled, so they should be disabled when you no longer need the encoder's data.

HB `enable_encoder: <void> (<int>)`

`void enable_encoder(int encoder)`

Enables the given encoder to start counting pulses and resets its counter to zero. By default encoders start in the disabled state and must be enabled before they start counting.

HB `exp10: float (<float>)`

`float exp10(float num)`

Returns 10 to the num power.

HB `exp: float (<float>)`

`float exp(float num)`

Returns *e* to the num power.

HB RCX `fd: <void> (<int>)`

`void fd(int m)`

Turns motor **`m`** on in the forward direction. Example: `fd(3);`

RCX `getticksremaining: <int> ()`

HB RCX `hog_processor: <void> ()`

`void hog_processor()`

Allocates an additional 256 milliseconds of execution to the currently running process. If this function is called repeatedly, the system will wedge and only execute the process that is calling `hog_processor()`. Only a system reset will unwedge from this state. Needless to say, this function should be used with extreme care, and should not be placed in a loop, unless wedging the machine is the desired outcome.

HB `init_expbd_servos: <int> (<int>)`

HB RCX `kill_process: void (<int>)`

`void kill_process(int pid);`

The `kill_process` function is used to destroy processes. Processes are destroyed by passing their process ID number to `kill_process`. If the return value is 0, then the process was destroyed. If the return value is 1, then the process was not found. The following code shows the `main` process creating a `check_sensor` process, and then destroying it one second later:

```
void main() {
   int pid;
   pid= start_process(check_sensor(2));
   sleep(1.0);
   kill_process(pid);
}
```

HB `knob: <int> ()`

`int knob()`

Returns a value from 0 to 255 based on the position of a potentiometer. On the 6.270 board, the potentiometer is labelled **`frob knob`**.

RCX `light: <int> (<int>)`

RCX `light_passive: <int> (<int>)`

HB `log10: float (<float>)`

`float log10(float num)`

Returns logarithm of num to the base 10.

HB `log: float (<float>)`

`float log(float num)`

Returns natural logarithm of num.

RCX `moreticks: <void> (<int>)`

HB RCX `motor: <void> (<int>, <int>)`

`void motor(int m, int p)`

Turns on motor **`m`** at power level **`p`**. Power levels range from 100 for full on forward to -100 for full on backward.

HB RCX `mseconds: long ()`

`long mseconds()`

Returns the count of system time in milliseconds. Time count is reset by hardware reset (i.e., pressing reset switch on board) or the function `reset_system_time()`. `mseconds()` is implemented as a C primitive (not as a library function).

HB RCX `msleep: <void> (<long>)`

`void msleep(long msec)`

Waits for an amount of time equal to or greater than msec milliseconds. msec is a long integer. Example:

```
/* wait for 1.5 seconds */ msleep(1500L);
```

HB RCX `off: <void> (<int>)`

`void off(int m)`

Turns off motor `m`. Example: `off(1);`

RCX `poweroff: <void> ()`

RCX `prgm_button: <int> ()`

HB RCX `printf: void (<char[]>, ...)`

LCD Screen Printing

IC has a version of the C function `printf` for formatted printing to the LCD screen. The syntax of `printf` is the following:

```
    printf(format-string, [arg-1] , ... , [arg-N] )
```

This is best illustrated by some examples.

Printing Examples

Example 1: Printing a message. The following statement prints a text string to the screen.

```
printf("Hello, world!\n");
```

In this example, the format string is simply printed to the screen. The character `\n` at the end of the string signifies *end-of-line*. When an end-of-line character is printed, the LCD screen will be cleared when a subsequent character is printed. Thus, most `printf` statements are terminated by a `\n`.

Example 2: Printing a number. The following statement prints the value of the integer variable x with a brief message.

```
printf("Value is %d\n", x);
```

The special form `%d` is used to format the printing of an integer in decimal format.

Example 3: Printing a number in binary. The following statement prints the value of the integer variable x as a binary number.

```
printf("Value is %b\n", x);
```

The special form `%b` is used to format the printing of an integer in binary format. Only the *low byte* of the number is printed.

Example 4: Printing a floating point number. The following statement prints the value of the floating point variable `n` as a floating point number.

```
printf("Value is %f\n", n);
```

The special form `%f` is used to format the printing of floating point number.

Example 5: Printing two numbers in hexadecimal format.

```
printf("A=%x  B=%x\n", a, b);
```

The form `%x` formats an integer to print in hexadecimal.

Formatting Command Summary

Format Command	Data Type	Description
`%d`	`int`	**decimal number**
`%x`	`int`	**hexadecimal number**
`%b`	`int`	**low byte as binary number**
`%c`	`int`	**low byte as ASCII character**
`%f`	`float`	**floating point number**
`%s`	`*char`	**character array (string)**

Special Notes

- The final character position of the LCD screen is used as a system "heartbeat." This character continuously blinks between a large and small heart when the board is operating properly. If the character stops blinking, the board has failed.
- Characters that would be printed beyond the final character position are truncated.
- When using a two-line display, the printf() command treats the display as a single longer line.
- Printing of long integers is not presently supported.

HB `random: <int> (<int>)`

HB `read_encoder: <int> (<int>)`

```
int read_encoder(int encoder)
```

Returns the number of pulses counted by the given encoder since it was enabled or since the last reset, whichever was more recent.

RCX `reset: <void> ()`

HB `reset_encoder: <void> (<int>)`

```
void reset_encoder(int encoder)
```

Resets the counter of the given encoder to zero. For an enabled encoder, it is more efficient to reset its value than to use `enable_encoder()` to clear it.

HB RCX `reset_system_time: <void> ()`

`void reset_system_time()`

Resets the count of system time to zero milliseconds.

HB `seconds: <float> ()`

`float seconds()`

Returns the count of system time in seconds, as a floating point number. Resolution is one millisecond.

HB RCX `set_beeper_pitch: <void> (<float>)`

`void set_beeper_pitch(float frequency)`

Sets the beeper tone to be `frequency` Hz. The subsequent function is then used to turn the beeper on.

HB `set_digital_out: <int> (<int>)`

RCX `setticks: <void> (<int>)`

HB RCX `sin: float (<float>)`

HB RCX `sleep: <void> (<float>)`

`void sleep(float sec)`

Waits for an amount of time equal to or slightly greater than `sec` seconds. `sec` is a floating point number. Example:

```
/* wait for 1.5 seconds */ sleep(1.5);
```

HB `sonar: <int> ()`

HB RCX `sqrt: float (<float>)`

`float sqrt(float num)`

Returns square root of `num`.

HB RCX `start_button: <int> ()`

`int start_button()`

Returns value of button labelled **`Start`** (or **`Escape`**). Example:

```
/* wait for button to be pressed; then wait for it to be released
so that button press is debounced */
while (!start_button()) {}
while (start_button()) {}
```

HB `start_press: <void> ()`

`void start_press()`

Like `stop_press()`, but for the **`Start`** button.

HB RCX `stop_button: <int> ()`

```
int stop_button()
```

Returns value of button labelled `Stop` (or `Choose`): 1 if pressed and 0 if released. Example:

```
/* wait until stop button pressed */
while (!stop_button()) {}
```

HB `stop_press: <void> ()`

```
void stop_press()
```

Waits for the `Stop` button to be pressed, then released. Then issues a short beep and returns. The code for `stop_press()` is as follows:

```
while (!stop_button()); while (stop_button()); beep();
```

HB `_system_print_off: <void> ()`
HB `_system_print_on: <void> ()`
HB `_system_pwm_off: <void> ()`
HB `_system_pwm_on: <void> ()`

HB RCX `tan: float (<float>)`

```
float tan(float angle)
```

Returns tangent of `angle`. Angle is specified in radians; result is in radians.

HB `test_digital_out: <int> (<int>)`

HB RCX `tone: <void> (<float>, <float>)`

```
void tone(float frequency, float length)
```

Produces a tone at pitch `frequency` Hertz for `length` seconds. Returns when the tone is finished. Both `frequency` and `length` are floats.

RCX `touch: <int> (<int>)`

RCX `view_button: <int> ()`