Appendix A

C Application Programming Interface (API)

This C API is assembled with use of the C reference card of Joseph H. Silverman, January 2003 v2.0, *copyright* ©2003 [11].

A.1 Program structure/functions

function declarations	*type* $fnc(type_1, \ldots)$
external variable declarations	*type name*
main routine	`main() {`
local variable declarations	*declarations*
	statements
	}
function definition	*type* $fnc(arg_1, \ldots)$ {
local variable declarations	*declarations*
	statements
	`return` *value*`;`
	}
comments	`/* */`
main with args	`main(int argc, char *argv[])`
terminate execution	`exit(`*arg*`)`

A.2 C Preprocessor

include library file	`#include` $< filename >$
include user file	`#include` "$filename$"
replacement text	`#define` *name text*
replacement macro	`#define` *name*(*var*) `text`
undefine	`#undef` *name*
is *name* defined, not defined?	`#ifdef, #ifndef`

A.3 Data Types/Declarations

character (1 byte)	`char`
integer	`int`

float (single precision)	`float`
float (double precision)	`double`
short (16 bit integer)	`short`
long (32 bit integer)	`long`
positive and negative	`signed`
only positive	`unsigned`
pointer to `int`, `float`, ...	`*int, *float, ...`
enumeration constant	`enum`
constant (unchanging) value	`const`
declare external variable	`extern`
local to source file	`static`
no value	`void`
structure	`struct`
create name by data type	`typedef` *typename*
size of an object (type is `size_t`)	`sizeof` *object*
size of a data type (type is `size_t`)	`sizeof` (*type name*)

A.4 Initialization

initialize variable	*type name* = *value*
initialize array	*type name*[ ] = {$value_1, \ldots$}
initialize char string	`char` *name*[ ] = "*string*"

A.5 Pointers, Arrays & Structures

declare pointer to *type*	*type* `*`*name*
declare function returning pointer to *type*	*type* `*f()`
declare pointer to function returning *type*	*type* `(*pf)()`
generic pointer type	`void *`
null pointer	`*`*pointer*
address of object *name*	`&`*name*
array	*name*`[`*dim*`]`
multi-dim array	*name*`[`dim_1`][`dim_2`]...`
Structures	
structure template	`struct` *tag* {
declaration of members	*declarations*
	};
create structure	`struct` *tag name*
member of structure from template	*name.member*

A.6 Operators

structure member operator	*name.member*
structure pointer	*pionter*`->`*member*
increment, decrement	`++, --`
plus, minus, logical not, bitwise not	`+, -, !,` $\sim$
indirection via pointer, address of object	`*`*pointer*`, &`*name*
cast expression to type	`(`*type*`)` *expr*
size of an object	`sizeof`
multiply, divide, modulus (remainder)	`*, /, %`
add, subtract	`+, -`

left, right shift [bit ops]	`<<, >>`
comparisons	`>, >=, <, <=, ==, !=`
bitwise and	`&`
bitwise exclusive or	∧
bitwise or (incl)	`|`
logical and	`&&`
logical or	`||`
conditional expression	$expr_1$ `?` $expr_2$ `:` $expr_3$
assignment operators	`+=, -=, *=, ...`
expression evaluation separator	`,`

A.7 Flow of control

statement terminator	`;`
block delimiters	{ }
exit from `switch, while, do, for`	`break`
next iteration of `while, do, for`	`continue`
go to	`goto` *label*
label	*label*`:`
return value from function	`return` *expr*
Flow constructions	
`if` statement	`if (`*expr*`)` *statement* `else if (`*expr*`)` *statement* `else` *statement*
`while` statement	`while (`*expr*`)` *statement*
`for` statement	`for (`$expr_1$`;` $expr_2$`;` $expr_3$`)` *statement*
`do` statement	`do` *statement* `while(`*expr*`);`

`switch` statement	`switch (`*expr*`) {`
	  `case` $const_1$`:` $statement_1$ `break;`
	  `case` $const_2$`:` $statement_2$ `break;`
	  `default:` *statement*
	`}`

A.8 Character Class Tests <ctype.h>

alphanumeric?	`isalnum(c)`
alphabetic?	`isalpha(c)`
control character?	`iscntrl(c)`
decimal digit?	`isdigit(c)`
printing character (not incl space)?	`isgraph(c)`
lower case letter?	`islower(c)`
printing character (incl space)?	`isprint(c)`
printing char except space, letter, digit?	`ispunct(c)`
space, formfeed, newline, cr, tab, vtab?	`isspace(c)`
upper case letter?	`isupper(c)`
hexadecimal digit?	`isxdigit(c)`

A.9 String Operations <string.h>

In the following functions, `s,t` denote strings and `cs,ct` denote constant strings.

length of `s`	`strlen(s)`
copy `ct` to `s`	`strcpy(s,ct)`
copy `ct` to `s` up to `n` chars	`strncpy(s,ct,n)`
concatenate `ct` after `s`	`strcat(s,ct)`
concatenate `ct` after `s` up to `n` chars	`strncat(s,ct,n)`
compare `cs` to `ct`	`strcmp(cs,ct)`
compare `cs` to `ct` only first `n` chars	`strncmp(cs,ct,n)`
pointer to first `c` in `cs`	`strchr(cs,c)`
pointer to last `c` in `cs` * strrchr(cs,c)	
copy `n` chars form `ct` to `s`	`memcpy(s,ct,n)`
copy `n` chars from `ct` to `s` (may overlap)	`memmove(s,ct,n)`
compare `n` chars of `cs` with `ct`	`memcmp(cs,ct,n)`
pointer to first `c` in first `n` chars of `cs`	`memchr(cs,c,n)`
put `c` into first `n` chars of `cs`	`memset(s,c,n)`

A.10 Input/Output <stdio.h>

Standard I/O	
standard input stream	`stdin`
standard output stream	`stdout`
standard error stream	`stderr`
end of file	`EOF`
get a character	`getchar()`
print a character	`putchar(`chr`)`
print formatted data	`printf("`$format$`",`arg_1`,...)`
print to string `s`	`sprintf(s,"`$format$`",`arg_1`,...)`
read formatted data	`scanf("`$format$`",&`$name_1$`,...)`
read from string `s`	`sscanf(s,"`$format$`",&`$name_1$`,...)`
read ling to string `s` (`< max` chars)	`gets(s,max)`
print string `s`	`puts(s)`
File I/O	
declare file pointer	`FILE *`fp
pointer to named file	`fopen("`$name$`","`$mode$`")`
modes: `r` (read), `w` (write), `a` (append)	
get a character	`getc(`fp`)`
write a character	`putc(`chr`,`fp`)`
write to file	`fprintf(`fp`,"`$format$`",`arg_1`,...)`
read from file	`fscanf(`fp`,"`$format$`",`arg_1`,...)`
close file	`fclose(`fp`)`
non-zero if error	`ferror(`fp`)`
non-zero if EOF	`feof(`fp`)`
read line to string `s` (`< max` chars)	`fgets(s,max,`fp`)`
write string `s`	`fputs(s,`fp`)`
Codes for formatted I/O	
left justify	`-`
print with sign	`+`
print space if no sign	*space*
pad with leading zeros	`0`
min field width	w

precision	*p*
conversion characters	*m*
short	`h`
long	`l`
long double	`L`
conversion characters	*c*
integer	`d,i`
single char	`c`
double	`f`
octal	`o`
pointer	`p`
unsigned	`u`
char string	`s`
exponential	`e,E`
hexadecimal	`x,X`
number of chars written	`n`

A.11 Standard Utility Functions <stdlib.h>

absolute value of `int n`	`abs(n)`
absolute value of `long n`	`labs(n)`
quotient and remainder of `ints n,d`	`div(n,d)`
returns structure with `div_t.quot` and `div_t.rem`	
quotient and remainder of `longs n,d`	`ldiv(n,d)`
returns structure with `ldiv_t.quot` and `ldiv_t.rem`	
pseudo-random integer `[0,RAND_MAX]`	`rand()`
set random seed to `n`	`srand(n)`
terminate program execution	`exit(status)`
pass string `s` to system for execution	`system(s)`
Conversions	
convert string `s` to double	`atof(s)`
convert string `s` to integer	`atoi(s)`
convert string `s` to long	`atol(s)`
convert prefix of `s` to double	`strtod(s,endp)`
convert prefix of `s` (base `b`) to long	`strtol(s,endp,b)`
convert prefix of `s`(base `b`) to unsigned long	`strtoul(s,endp,b)`
Storage Allocation	
allocate storage	`malloc(size), calloc(nobj,size)`
change size of object	`realloc(pts,size)`
deallocate space	`free(prt)`
Array Functions	
search `array` for `key`	`bsearch(key,array,n,size,cmp())`
search `array` ascending order	`qsort(array,n,size,cmp())`

A.12 Time and Date Functions <time.h>

processor time used by program	`clock()`
time in seconds	`clock()/CLOCKS_PER_SEC`
current calender time	`time()`
$\texttt{time}_2\texttt{-time}_2$ in seconds (double)	$\texttt{difftime(time}_2\texttt{,time}_1\texttt{)}$
arithmetic types representing times	`clock_t,time_t`

A.13 Mathematical Functions <math.h>

The arguments and returned values of the mathematical functions are all of type `double`.

trigonometric functions	`sin(x), cos(x), tan(x)`
inverse trigonometric functions	`asin(x), acos(x), atan(x)`
arctan(y/x)	`atan2(y,x)`
hyperbolic trigonometric functions	`sinh(x), cosh(x), tanh(x)`
exponentials & logs	`exp(x), log(x), log10(x)`
exponentials & logs (2 power)	`ldexp(x,n), frexp(x,*e)`
division & remainder	`modf(x,*ip), fmod(x,y)`
powers	`pow(x,y), sqrt(x)`
rounding	`ceil(x), floor(x), fabs(x)`

A.14 Integer Type Limits <limits.h>

bits in `char`	`CHAR_BIT`
max value of `char`	`CHAR_MAX`
min value of `char`	`CHAR_MIN`
max value of `int`	`INT_MAX`
min value of `int`	`INT_MIN`
max value of `long`	`LONG_MAX`
min value of `long`	`LONG_MIN`
max value of `signed char`	`SCHAR_MAX`
min value of `signed char`	`SCHAR_MIN`
max value of `short`	`SHRT_MAX`
min value of `short`	`SHRT_MIN`
max value of `unsigned char`	`UCHAR_MAX`
max value of `unsigned int`	`UINT_MAX`
max value of `unsigned long`	`ULONG_MAX`
max value of `unsigned short`	`USHRT_MAX`

A.15 Float Type Limits <float.h>

radix of exponent rep	`FLT_RADIX`
floating point rounding mode	`FLT_ROUNDS`
decimal digits of precision	`FLT_DIG`
smallest x so $1.0 + x \neq 1.0$	`FLT_EPSILON`
maximum floating point number	`FLT_MAX`
maximum exponent	`FLT_MAX_EXP`
minimum floating point number	`FLT_MIN`
minimum exponent	`FLT_MIN_EXP`
decimal digits of precision	`DBL_DIG`
smallest x so $1.0 + x \neq 1.0$	`DBL_EPSILON`
maximum `double` floating point number	`DBL_MAX`
maximum exponent	`DBL_MAX_EXP`
minimum `double` floating point number	`DBL_MIN`
minimum exponent	`DBL_MIN_EXP`

Appendix B

BrickOS Application Programming Interface (API)

In this appendix, a subset of the complete BrickOS API [9] is given. The functions mentioned are selected on their relevance for the course goal.

B.1 Console input/output `<conio.h>`

The console input/output functions are contained in the header file *include/conio.h*. The digit display positions are numerated from right to left, starting with 0 for the digit to the right of the running man.

LCD positions: 5 4 3 2 1 {man} 0

NOTE: position 5 is only partially present on the LCD display.

Functions

void **delay** (unsigned ms)
    delay approximately ms mSec

void **cputc_native_*pos*** (char mask)
    write bit-pattern for segments at position pos of LCD. The position pos can vary from 0 up to 5

void **cputc_native** (char mask, int pos)
    Set/Clear individual segments at specified position pos of LCD

void **cputc_hex_*pos*** (unsigned nibble)
    write HEX digit to position pos of LCD

void **cputc_hex** (char c, int pos)
    write HEX digit to specific position pos of LCD

void **cputw** (unsigned word)
    write a HEX word to LCD

void **cputc_*pos*** (unsigned c)
    write ASCII char to position pos of LCD

void **cputs** (char *s)

Write string s to LCD (Only first 5 chars)

void **cls** ()
clear user portion of LCD

B.2 Direct control of LCD display <dlcd.h>

The direct control of LCD display interface is present in the header file *include/dlcd.h*.

Defines

#define **dlcd_show**(a) bit_set(BYTE_OF(a),BIT_OF(a))
set a segment directly in the LCD buffer

#define **dlcd_hide**(a) bit_clear(BYTE_OF(a),BIT_OF(a))
clear a segment directly in the LCD buffer

#define **dlcd_store**(a) bit_store(BYTE_OF(a),BIT_OF(a))
store the carry flag to a segment directly in the LCD buffer

#define **BYTE_OF**(a, b) a
helper macros

#define **BIT_OF**(a, b) b

B.3 Direct motor control <dmotor.h>

The direct motor control functions are contained in the header file *include/dmotor.h*.

Defines

#define **MIN_SPEED** 0
minimum motor speed

#define **MAX_SPEED** 255
maximum motor speed

Enumerations

enum **MotorDirection** { **off** = 0, **fwd** = 1, **rev** = 2, **brake** = 3 }
the motor directions

The Enumeration values are

off	freewheel
fwd	forward
rev	reverse
brake	hold current position

Functions

void **motor_*mot*_dir** (**MotorDirection** dir)
set motor mot direction to dir

void **motor_*mot*_speed** (unsigned char speed)

set motor mot speed

Variables

const unsigned char **dm_*mot*_pattern** [4]
motor mot drive patterns

Motorstate **dm_*mot***
motor mot state

B.4 Direct reading of sensors <dsensor.h>

The direct reading of sensors functions are contained in the header file *include/dsensor.h*.

Defines

#define **SENSOR_1** AD_C
Sensor on input pad 1

#define **SENSOR_2** AD_B
Sensor on input pad 2

#define **SENSOR_3** AD_A
Sensor on input pad 3

#define **BATTERY** AD_D
Battery sensor

#define **LIGHT_RAW_BLACK** 0xffc0
activate light sensor raw black value

#define **LIGHT_RAW_WHITE** 0x5080
activate light sensor raw white value

#define **LIGHT**(a) (147 - ds_scale(a)/7)
map light sensor to 0...LIGHT_MAX

#define **LIGHT_MAX** LIGHT(LIGHT_RAW_WHITE)
maximum decoded value

#define **LIGHT_*sen*** LIGHT(SENSOR_*sen*)
light sensor on input sen

#define **ROTATION_*sen*** (**ds_rotations**[*sen*])
rotation sensor on input sen

#define **TOUCH**(a) ((unsigned int)(a) < 0x8000)
convert raw data to touch sensor (0: off, else pressed)

#define **TOUCH_*sen*** TOUCH(SENSOR_*sen*)
touch sensor on input sen

#define **ds_scale**(x) ((unsigned int)(x)>>6)
mask off bottom 6 bits

#define **ds_unscale**(x) ((unsigned int)(x)<<6)

leave room for bottom 6 bits

Functions

void **ds_active** (volatile unsigned *sensor)
set sensor mode to active (light sensor emits light, rotation works)

void **ds_passive** (volatile unsigned *sensor)
set sensor mode to passive (light sensor detects ambient light)

void **ds_rotation_set** (volatile unsigned *sensor, int pos)
set rotation to an absolute value

void **ds_rotation_on** (volatile unsigned *sensor)
start tracking rotation sensor

void **ds_rotation_off** (volatile unsigned *sensor)
stop tracking rotation sensor

Variables

unsigned char **ds_activation**
activation bitmask

unsigned char **ds_rotation**
rotation bitmask

volatile int **ds_rotations** [3]
rotational position

B.5 Direct control of sound `<dsound.h>`

The direct sound control is included in the header file *include/dsound.h*.

Data Structures

struct **note_t**
the note structure describing a single note

Defines

#define **PITCH_PAUSE** 97
specify a pause (rest)

#define **PITCH_MAX** 98
maximum pitch value

#define **PITCH_END** 255
mark the end of a list of **note_t** entries

#define **DSOUND_BEEP** 0
system sounds

#define **DSOUND_SYS_MAX** 1
max system sound

#define **DSOUND_DEFAULT_16th_ms** 200
default duration of 1/16th note in ms

#define **DSOUND_DEFAULT_internote_ms** 200
default duration inter-note spacing in ms

Functions

void **dsound_play** (const **note_t** *notes)
play a sequence of notes

void **dsound_system** (unsigned nr)
play a system sound

unsigned **dsound_set_duration** (unsigned duration)
set duration of a 16th note in ms; return the previous duration

void **dsound_set_internote** (unsigned duration)
set duration of a inter-note spacing (subtracted from note duration)

int **dsound_playing** (void)
returns nonzero value if a sound is playing

wakeup_t dsound_finished (**wakeup_t** data)
sound finished event wakeup function

void **dsound_stop** (void)
stop playing sound

B.6 Memory data types <mem.h>

The memory data types are contained in the header file *include/mem.h*.

Defines

#define **NULL** ((void*)0)
null pointer value

Typedefs

typedef unsigned **size_t**
data type for memory sizes

B.7 Semaphores for task synchronization <semaphore.h>

The task synchronization semaphores are included in the header file *include/semaphore.h*.

Defines

#define **EAGAIN** 0xffff
the error code

Typedefs

typedef **atomic_t sem_t**
the semaphore data-type

Functions

int **sem_init** (**sem_t** *sem, int pshared, unsigned int value)
initialize a semaphore sem with initial value for count, the argument pshared is ignored

int **sem_wait** (**sem_t** *sem)
wait for semaphore (blocking)

int **sem_timedwait** (**sem_t** *sem, const **time_t** abs_timeout)
wait for semaphore (blocking with timeout) where abs_timeout denotes the absolute timeout of this operation. If the semaphore cannot be locked up to this time, this function returns -1, otherwise 0 is returned

int **sem_trywait** (**sem_t** *sem)
try a wait for semaphore (non-blocking)

int **sem_post** (**sem_t** *sem)
post a semaphore

int **sem_getvalue** (**sem_t** *sem, int *sval)
get the semaphore value

int **sem_destroy** (**sem_t** *sem)
we're done with the semaphore, destroy it

B.8 Reduced standard C library `<stdlib.h>`

The reduced standard C library is contained in the header file *include/stdlib.h*. The functions of this header file describe the public programming interface for memory management services and random number services.

Functions

void * **calloc**(**size_t** nmemb, **size_t** size)
allocate and return pointer to initialized memory
calloc() allocates memory for an array of {nmemb} elements of {size} bytes each and returns a pointer to the allocated memory. The memory is filled with zero values.

void * **malloc**(**size_t** size)
allocate and return pointer to uninitialized memory

void **free** (void *ptr)
return the allocated memory to memory management
free() frees the memory space pointed to by {ptr}, which must have been returned by a previous call to **malloc()**, or **calloc()**. Otherwise, or if free(ptr) has already been called before, undefined behavior occurs. If ptr is NULL, no operation is performed

long int **random** (void)
generate a random number in the range from 0 to RAND_MAX

void **srandom** (unsigned int seed)
seed the random number generator, default value 1

B.9 String functions <string.h>

The string functions are contained in the header file *inlude/string.h*.

Functions

void * **memcpy** (void *dest, const void *src, **size_t** size)
copy memory block of size bytes from source src to destination dest

void * **memset** (void *s, int c, **size_t** n)
fill memory block of n bytes with a byte value c from start s

char * **strcpy** (char *dest, const char *src)
copy null-terminated string from src to dest

int **strlen** (const char *s)
determine string length

int **strcmp** (const char *s1, const char *s2)
compare two strings

B.10 Time-related data and types <time.h>

The time-related data and types are contained in the header file *include/time.h*.

Defines

#define **TICK_IN_MS** 1
timer tick in ms

#define **TICK_PER_SEC** 1000
number of ms ticks in 1 sec

#define **SECS_TO_TICK**(a) ((a)*TICKS_PER_SEC)
conv. sec's to ticks

#define **MSECS_TO_TICK**(a) ((a)/TICKS_IN_MS)
conv. msec's to ticks

Typedefs

typedef unsigned long **time_t**
time type

Functions

time_t get_system_up_time (void)

B.11 Task management <tm.h>

The task management functions are contained in the header file *include/tm.h*.

Defines

#define **PRIO_LOWEST** 1
: the lowest possible task priority

#define **PRIO_NORMAL** 10
: the priority of most tasks

#define **PRIO_HIGHEST** 20
: the highest possible task priority

#define **T_DEAD** 0
: dead and gone, stack freed

#define **T_ZOMBIE** 1
: terminated, cleanup pending

#define **T_WAITING** 2
: waiting for an event

#define **T_SLEEPING** 3
: sleeping, wants to run

#define **T_RUNNING** 4
: running

#define **T_KERNEL** $(1 \ll 0)$
: kernel task

#define **T_USER** $(1 \ll 1)$
: user task

#define **T_IDLE** $(1 \ll 2)$
: idle task

#define **T_SHUTDOWN** $(1 \ll 7)$
: shutdown requested

#define **DEFAULT_STACK_SIZE** 512
: that's enough

#define **shutdown_requested()** ((**ctid**->tflags & T_SHUTDOWN) != 0)
: test to see if task has been asked to shutdown. If set, the task should shutdown as soon as possible. If clear, continue running.

Typedefs

typedef volatile unsigned char **tstate_t**
: task state type

typedef volatile unsigned char **tflags_t**
: task flags type

typedef unsigned char **priority_t**
: task priority type

typedef unsigned long **wakeup_t**

wakeup data area type

typedef signed int **tid_t**
task id type

Variables

tdata_t * **ctid**

B.12 Reduced UNIX standard library <unistd.h>

The reduced UNIX standard library functions are contained in the header file *include/unistd.h*.

Functions

tid_t execi (int(*code_start)(int, char **), int argc, char **argv, **priority_t** priority, **size_t** stack_size)
starts executing called from user code with parameters: *code_start* the entry-point of the new task, *argc* the count of arguments passed (0 if none), *argv* an array of pointers each pointing to an argument (NULL if none), *priority* the priority at which to run this task and *stack_size* the amount of memory in bytes to allocate to this task for its call stack. Returns -1 if failed to start, else tid (task-id)

void **shutdown_task** (**tid_t** tid)
signal shutdown for a task

void **shutdown_tasks** (**tflags_t** flags)
signal shutdown for many tasks

void **kill** (**tid_t** tid)
kill specified (tid) task

void **killall** (**priority_t** p)
kill all tasks with priority less than or equal to p, excluding self

void **exit** (int code) __attribute__((noreturn))
exit task, returning code

void **yield** (void)
current task yields the rest of timeslice

wakeup_t wait_event (**wakeup_t**(*wakeup)(**wakeup_t**), **wakeup_t** data)
suspend task until wakeup function returns non-null with parameters *wakeup* the function to be called when woken up and *data* the wakeup_t structure to be passed to the called function. Returns the wakeup() return value

unsinged int **sleep** (unsigned int sec)
delay execution allowing other tasks to run

unsigned int **msleep** (unsigned int msec)
delay execution allowing other tasks to run

B.13 Link networking protocol <lnp/lnp.h>

The LNP interface (link networking protocol) can be found in the header file *include/lnp/lnp.h*.

Defines

#define **LNP_DUMMY_INTEGRETY** ((**lnp_integrety_handler_t**)0)
dummy integrity layer packet handler

#define **LNP_DUMMY_ADDRESSING** ((**lnp_addressing_handler_t**)0)
dummy addressing layer packet handler

#define **LNP_DUMMY_REMOTE** ((**lnp_remote_handler_t**)0)
dummy remote packet handler

Typedefs

typedef void(* **lnp_integrity_handler_t**)(const unsigned char *, unsigned char)
the integrity layer packet handler type

typedef void(* **lnp_addressing_handler_t**)(const unsigned char *, unsigned char, unsigned char)
the addressing layer packet handler type

typedef void(* **lnp_remote_handler_t**)(unsigned int)
handler for remote

Functions

void **lnp_integrity_set_handler** (**lnp_integrity_handler_t** handler)
set the integrity layer packet handler

void **lnp_addressing_set_handler** (unsigned char port, **lnp_addressing_handler_t** handler)
set an addressing layer packet handler for a port

void **lnp_set_hostaddr** (unsigned char host)
set new LNP host address

void **lnp_remote_set_handler** (**lnp_remote_handler_t** handler)
set the remote packet handler

int **send_msg** (unsigned char msg)
send a standard firmware message

void **clear_msg** (void)
clear last message from standard firmware

wakeup_t msg_received (**wakeup_t** m)
wait until receive a message

unsigned char **get_msg** (void)
read received message from standard firmware

int **lnp_integrity_write** (const unsigned char *data, unsigned char length)
send a LNP integrity layer packet of given length

int **lnp_addressing_write** (const unsigned char *data, unsigned char length,
unsigned char dest, unsigned char srcport)
send a LNP addressing layer packet of given length

Variables

volatile **lnp_integrity_handler_t lnp_integrity_handler**
there are no ports for integrity layer packets, so there's just

volatile **lnp_addressing_handler_t lnp_addressing_handler** []
addressing layer packets may be directed to a variety of ports

unsigned char **lnp_hostaddr**
LNP host address

lnp_remote_handler_t lnp_remote_handler
packets from remote have no ports

unsigned char **lnp_rcx_message**
message variable

B.14 Link networking protocol logical layer <lnp/lnp-logical.h>

The LNP networking protocol logical layer functions can be found in the header file *include/lnp/lnp-logical.h*.

Functions

void **lnp_logical_range** (int far)
set the IR transmitter range, *far* 0 sets short range, *far* 1 sets long range

int **lnp_logical_range_is_far** (void)
test the IR transmitter range setting

int **lnp_logical_write** (const void *buf, **size_t** len)
write buffer to IR port, *buf* is a pointer to the array to be written, *len* determines the number of chars to be written in the array

void **lnp_logical_fflush** (void)
empty the IR receive buffer

B.15 RCX LCD control <rom/lcd.h>

The RCX LCD control is present in the header file *include/rom/lcd.h*.

Defines

#define **lcd_int**(i) lcd_number(i,sign,e0)
display an integer in decimal

#define **lcd_unsigned**(u) lcd_number(u,unsign,e0)
display an unsigned value in decimal

#define **lcd_clock**(t) lcd_number(t,unsign,e_2)
display a clock

#define **lcd_digit**(d) lcd_number(d,digit,digit_comma)
display a single digit right of the man symbol

Functions

void **lcd_show** (**lcd_segment** segment)
show LCD segment

void **lcd_hide** (**lcd_segment** segment)
hide LCD segment

void **lcd_clear** (void)
clear LCD display