For Rabbit Semiconductor Microprocessors

Integrated C Development System

Function Reference Manual

019-0113 • 020409 - A

SE and Premier Editions

Table of Contents

Dynamic C Functions Listed by Group

Floating-Point Math

Global Positioning System

HDLC Protocol (Rabbit 3000)

I/O

I2C Protocol

Interrupts

Low-Level Flash Access

MD5

MicroC/OS-II

Miscellaneous

Multitasking

Number-to-String Conversion

Pulse Width Modulation (Rabbit 3000)

Quadrature Decoder (Rabbit 3000)

Real-Time Clock

Serial Communication

Serial Packet Driver

SPI

Stdio

String Manipulation

String-to-Number Conversion

System

User ID Block

Watchdogs

Dynamic C Functions Listed Alphabetically

Symbols

A

B

C

D

E

F

G

H

I

K

L

M

O

P

Q

R

S

T

U

V

W

X

1. Function Descriptions

abs

```
int abs(int x);
```

DESCRIPTION

Computes the absolute value of an integer argument.

PARAMETERS

x Interger argument

RETURN VALUE

Absolute value of the argument.

LIBRARY

MATH.LIB

SEE ALSO

fabs

acos

```
float acos(float x);
```

DESCRIPTION

Computes the arccosine of real **float** value **x**.

PARAMETERS

x Assumed to be between -1 and 1.

RETURN VALUE

Arccosine of the argument.
If **x** is out of bounds, the function returns **0** and signals a domain error.

LIBRARY

MATH.LIB

SEE ALSO

cos, cosh, asin, atan

acot

```
float acot(float x);
```

DESCRIPTION

Computes the arcotangent of real **float** value **x**.

PARAMETERS

x Assumed to be between -INF and +INF.

RETURN VALUE

Arccotangent of the argument.

LIBRARY

MATH.LIB

SEE ALSO

tan, atan

acsc

```
float acsc(float x);
```

DESCRIPTION

Computes the arccosecant of real **float** value **x**.

PARAMETERS

x Assumed to be between -INF and +INF.

RETURN VALUE

The arccosecant of the argument.

LIBRARY

MATH.LIB

SEE ALSO

sin, asin

AESdecrypt

```
void AESdecrypt(char *data, char *expandedkey, int nb, int nk );
```

DESCRIPTION

Decrypts a block of data using an implementation of the Rijndael AES cipher. The encryped block of data is overwritten by the decrypted block of data.

PARAMETERS

data A block of data to be decrypted.

expandedkey A set of round keys (generated by **AESexpandKey()**).

nb The block size to use. Block is 4 * **nb** bytes long.

nk The key size to use. Cipher key is 4 * **nk** bytes long.

LIBRARY

AES_CRYPT.LIB

AESdecryptStream

```
void AESdecryptStream(AESstreamState *state, char *data, int
  count);
```

DESCRIPTION

Decrypts an array of bytes using the Rabbit implementation of cipher feedback mode. See **Samples\Crypt\AES_STREAMTEST.C** for a sample program and a detailed explanation of the encryption/decryption process.

PARAMETERS

state The **AESstreamState** structure. This memory must be allocated in the program code before calling **AESdecrptyStream()**:

```
static AESstreamState decrypt_state;
```

data An array of bytes that will be decrypted in place.

count Size of data array

LIBRARY

AES_CRYPT.LIB

AESencrypt

```
void AESencrypt(char *data, char *expandedkey, int nb, int nk );
```

DESCRIPTION

Encrypts a block of data using an implementation of the Rijndael AES cipher. The block of data is overwritten by the encrypted block of data.

PARAMETERS

data	A block of data to be encrypted
expandedkey	A set of round keys (generated by **AESexpandKey()**)
nb	The block size to use. Block is 4 * **nb** bytes long
nk	The key size to use. Cipher key is 4 * **nk** bytes long

RETURN VALUE

None.

LIBRARY

AES_CRYPT.LIB

AESencryptStream

```
void AESencryptStream(AESstreamState *state, char *data, int
   count);
```

DESCRIPTION

Encrypts an array of bytes using the Rabbit implementation of cipher feedback mode. See **Samples\Crypt\AES_STREAMTEST.C** for a sample program and a detailed explanation of the encryption/decryption process.

PARAMETERS

state	The **AESstreamState** structure. This memory must be allocated in the program code before calling **AESencrptyStream()**: `static AESstreamState encrypt_state;`
data	An array of bytes that will be encrypted in place.
count	Size of data array.

LIBRARY

AES_CRYPT.LIB

AESexpandKey

```
void AESexpandKey(char *expanded, char *key, int nb, int nk, int
   rounds);
```

DESCRIPTION

Prepares a key for use by expanding it into a set of round keys. A key is a “password” to decipher encoded data.

PARAMETERS

expanded	A buffer for storing the expanded key. The size of the expanded key is 4 * **nb** * (**rounds** + 1).
key	The cipher key, the size should be 4 * **nk**
nb	The block size will be 4 * **nb** bytes long.
nk	The key size will be 4 * **nk** bytes long.
rounds	The number of cipher rounds to use.

RETURN VALUE

None.

LIBRARY

AES_CRYPT.LIB

AESinitStream

```
void AESinitStream(AESstreamState *state, char *key, char
   *init_vector);
```

DESCRIPTION

Sets up **AESstreamState** to begin encrypting or decrypting a stream. Each **AESstreamState** structure can only be used for one direction. See **Samples\Crypt\AES_STREAMTEST.C** for a sample program and a detailed explanation of the encryption/decryption process.

PARAMETERS

state	An **AESstreamState** structure to be initialized. This memory must be allocated in the program code before calling **AESinitStream()**.
key	The 16-byte cipher key, using a **NULL** pointer will prevent an existing key from being recalculated.
init_vector	A 16-byte array representing the initial state of the feedback registers. Both ends of the stream must begin with the same initialization vector.

RETURN VALUE

None.

LIBRARY

AES_CRYPT.LIB

asec

```
float asec(float x);
```

DESCRIPTION

Computes the arcsecant of real **float** value **x**.

PARAMETERS

x	Assumed to be between -INF and +INF.

RETURN VALUE

The arcsecant of the argument.

LIBRARY

MATH.LIB

SEE ALSO

cos, acos

asin

```
float asin(float x);
```

DESCRIPTION

Computes the arcsine of real **float** value **x**.

PARAMETERS

x	Assumed to be between -1 and +1.

RETURN VALUE

The arcsine of the argument.

LIBRARY

MATH.LIB

SEE ALSO

sin, acsc

atan

```
float atan(float x);
```

DESCRIPTION

Computes the arctangent of real **float** value **x**.

PARAMETERS

x	Assumed to be between -INF and +INF.

RETURN VALUE

The arctangent of the argument.

LIBRARY

MATH.LIB

SEE ALSO

tan, acot

atan2

```
float atan2(float y, float x);
```

DESCRIPTION

Computes the arctangent of real **float** value **y/x** to find the angle in radians between the x-axis and the ray through (0,0) and (x,y).

PARAMETERS

y	The point corresponding to the y-axis
x	The point corresponding to the x-axis

RETURN VALUE

If both **y** and **x** are zero, the function returns **0** and signals a domain error. Otherwise the arctangent of **y/x** is returned as follows:

Returned Value	Parameter Values
angle	x ≠ 0, y ≠ 0
PI/2	x = 0, y > 0
–PI/2	x = 0, y <
0	x > 0, y = 0
PI	x < 0, y = 0

LIBRARY

MATH.LIB

SEE ALSO

acos, asin, atan, cos, sin, tan

atof

```
float atof(char *sptr);
```

DESCRIPTION

ANSI String to Float Conversion (UNIX compatible).

PARAMETERS

sptr String to convert.

RETURN VALUE

The converted floating value.
If the conversion is invalid, **_xtoxErr** is set to **1**. Otherwise **_xtoxErr** is set to **0**.

LIBRARY

STRING.LIB

SEE ALSO

atoi, atol, strtod

atoi

```
int atoi(char *sptr);
```

DESCRIPTION

ANSI String to Integer Conversion (UNIX compatible).

PARAMETERS

sptr String to convert.

RETURN VALUE

The converted integer value.

LIBRARY

STRING.LIB

SEE ALSO

atol, atof, strtod

atol

```
long atol(char *sptr);
```

DESCRIPTION

ANSI String to Long Conversion (UNIX compatible).

PARAMETERS

sptr	String to convert.

RETURN VALUE

The converted long integer value.

LIBRARY

STRING.LIB

SEE ALSO

atoi, atof, strtod

bit

```
unsigned int bit(void *address, unsigned int bit);
```

DESCRIPTION

Dynamic C may expand this call inline

Reads specified bit at memory address. **bit** may be from 0 to 31. This is equivalent to the following expression, but more efficient: **(*(long *)address >> bit) & 1**

PARAMETERS

address	Address of byte containing bits 7-0
bit	Bit location where 0 represents the least significant bit

RETURN VALUE

1: Specified bit is set.
0: Bit is clear.

LIBRARY

UTIL.LIB

SEE ALSO

BIT

BIT

```
unsigned int BIT(void *address, unsigned int bit);
```

DESCRIPTION

Dynamic C may expand this call inline

Reads specified bit at memory address. **bit** may be from 0 to 31. This is equivalent to the following expression, but more efficient: **(*(long *)address>>bit) &1**

PARAMETERS

address	Address of byte containing bits 7-0
bit	Bit location where 0 represents the least significant bit

RETURN VALUE

1 if specified bit is set; **0** if bit is clear.

LIBRARY

UTIL.LIB

SEE ALSO

bit

BitRdPortE

```
root int BitRdPortE(unsigned int port, int bitnumber);
```

DESCRIPTION

Returns 1 or 0 matching the value of the bit read from the specified external I/O port.

PARAMETERS

port	Address of external parallel port data register.
bitnumber	Bit to read (0–7).

RETURN VALUE

0 or **1**: The value of the bit read.

LIBRARY

SYSIO.LIB

SEE ALSO

RdPortI, BitRdPortI, WrPortI, BitWrPortI, RdPortE, WrPortE, BitWrPortE

BitRdPortI

```
int BitRdPortI(int port, int bitnumber);
```

DESCRIPTION

Returns 1 or 0 matching the value of the bit read from the specified internal I/O port.

PARAMETERS

port	Address of internal parallel port data register.
bitnumber	Bit to read (0–7).

RETURN VALUE

0 or **1**: The value of the bit read.

LIBRARY

SYSIO.LIB

SEE ALSO

RdPortI, WrPortI, BitWrPortI, BitRdPortE, RdPortE, WrPortE, BitWrPortE

BitWrPortE

```
void BitWrPortE( unsigned int port, char *portshadow, int value,
   int bitcode);
```

DESCRIPTION

Updates shadow register at **bitcode** with **value** (0 or 1) and copies shadow to register.

WARNING! A shadow register is required for this function.

PARAMETERS

port	Address of external parallel port data register.
portshadow	Reference pointer to a variable to shadow the current value of the register.
value	Value of 0 or 1 to be written to the bit position.
bitcode	Bit position 0–7.

LIBRARY

SYSIO.LIB

SEE ALSO

RdPortI, BitRdPortI, WrPortI, BitWrPortI, BitRdPortE, RdPortE, WrPortE

BitWrPortI

```
void BitWrPortI( int port, char *portshadow, int value, int
   bitcode);
```

DESCRIPTION

Updates shadow register at position **bitcode** with **value** (0 or 1); copies shadow to register.

WARNING! A shadow register is required for this function.

PARAMETERS

port	Address of internal parallel port data register.
portshadow	Reference pointer to a variable to shadow the current value of the register.
value	Value of 0 or 1 to be written to the bit position.
bitcode	Bit position 0–7.

LIBRARY

SYSIO.LIB

SEE ALSO

RdPortI, BitRdPortI, WrPortI, BitRdPortE, RdPortE, WrPortE, BitWrPortE

ceil

```
float ceil(float x);
```

DESCRIPTION

Computes the smallest integer greater than or equal to the given number.

PARAMETERS

x Number to round up.

RETURN VALUE

The rounded up number.

LIBRARY

MATH.LIB

SEE ALSO

floor, fmod

chkHardReset

```
int chkHardReset( void );
```

DESCRIPTION

This function determines whether this restart of the board is due to a hardware reset. Asserting the RESET line or recycling power are both considered hardware resets. A watchdog timeout is not a hardware reset.

RETURN VALUE

1: The processor was restarted due to a hardware reset.
0: If it was not.

LIBRARY

Sys.lib

SEE ALSO

chkSoftReset, chkWDTO, _sysIsSoftReset

chkSoftReset

```
int chkSoftReset( void );
```

DESCRIPTION

This function determines whether this restart of the board is due to a software reset from Dynamic C or a call to **forceSoftReset()**.

RETURN VALUE

1: The board was restarted due to a soft reset.
0: If it was not.

LIBRARY

Sys.lib

SEE ALSO

chkHardReset, chkWDTO, _sysIsSoftReset

chkWDTO

```
int chkWDTO( void );
```

DESCRIPTION

This function determines whether this restart of the board is due to a watchdog timeout.

RETURN VALUE

1: If the board was restarted due to a watchdog timeout.
0: If it was not.

LIBRARY

Sys.lib

SEE ALSO

chkHardReset, chkSoftReset, _sysIsSoftReset

clockDoublerOn

```
void clockDoublerOn();
```

DESCRIPTION

Enables the Rabbit clock doubler. If the doubler is already enabled, there will be no effect. Also attempts to adjust the communication rate between Dynamic C and the board to compensate for the frequency change. User serial port rates need to be adjusted accordingly. Also note that single-stepping through this routine will cause Dynamic C to lose communication with the target.

LIBRARY

SYS.LIB

SEE ALSO

clockDoublerOff

clockDoublerOff

```
void clockDoublerOff();
```

DESCRIPTION

Disables the Rabbit clock doubler. If the doubler is already disabled, there will be no effect. Also attempts to adjust the communication rate between Dynamic C and the board to compensate for the frequency change. User serial port rates need to be adjusted accordingly. Also note that single-stepping through this routine will cause Dynamic C to lose communication with the target.

LIBRARY

SYS.LIB

SEE ALSO

clockDoublerOn

CoBegin

```
void CoBegin(CoData *p);
```

DESCRIPTION

Initialize a costatement structure so the costatement will be executed next time it is encountered.

PARAMETERS

p	Address of costatement

LIBRARY

COSTATE.LIB

cof_pktXreceive

```
int cof_pktXreceive(void *buffer, int buffer_size); X=A|B|C|D
```

DESCRIPTION

Receives an incoming packet. This function will return after a complete packet has been read into the buffer.

Starting with Dynamic C version 7.25, the functions **cof_pktEreceive()** and **cof_pktFreceive()** are available when using the Rabbit 3000 microprocessor.

PARAMETERS

buffer	A buffer for the packet to be written into.
buffer_size	Length of the buffer.

RETURN VALUE

>0: The number of bytes in the received packet on success.
0: No new packets have been received.
-1: The packet is too large for the given buffer.
-2: A needed **test_packet** function is not defined.

LIBRARY

PACKET.LIB

cof_pktXsend

```
void cof_pktXsend(void *send_buffer int buffer_length, char
   delay); X=A|B|C|D
```

DESCRIPTION

Initiates the sending of a packet of data. The function will exit when the packet is finished transmitting.

Starting with Dynamic C version 7.25, the functions **cof_pktEsend()** and **cof_pktFsend()** are available when using the Rabbit 3000 microprocessor.

PARAMETERS

send_buffer	The data to be sent.
buffer_length	Length of the data buffer to transmit.
delay	The number of byte times (0-255) to delay before sending data. This is used to implement protocol-specific delays between packets.

LIBRARY

PACKET.LIB

cof_serXgetc

```
int cof_serXgetc(); /* where X = A|B|C|D */
```

DESCRIPTION

This single-user cofunction yields to other tasks until a character is read from port X. This function only returns when a character is successfully written. It is non-reentrant.

Starting with Dynamic C version 7.25, the functions **cof_serEgetc()** and **cof_serFgetc()** may beused with the Rabbit 3000 microprocessor.

RETURN VALUE

An integer with the character read into the low byte.

LIBRARY

RS232.LIB

EXAMPLE

```
// echoes characters
main() {
   int c;
   serXopen(19200);
   loopinit();
   while (1) {
      loophead();
      wfd c = cof_serAgetc();
      wfd cof_serAputc(c);
   }
   serAclose();
}
```

cof_serXgets

```
int cof_serXgets(char *s, int max, unsigned long tmout);
   /* where X = A|B|C|D */
```

DESCRIPTION

This single-user cofunction reads characters from port X until a **NULL** terminator, line-feed, or carriage return character is read, **max** characters are read, or until **tmout** milliseconds transpires between characters read. A timeout will never occur if no characters have been received. This function is non-reentrant.

It yields to other tasks for as long as the input buffer is locked or whenever the buffer becomes empty as characters are read. **s** will always be **NULL** terminated upon return.

Starting with Dynamic C version 7.25, the functions **cof_serEgets()** and **cof_serFgets()** may be used with the Rabbit 3000 microprocessor.

PARAMETERS

s	Character array into which a **NULL** terminated string is read.
max	The maximum number of characters to read into s.
tmout	Millisecond wait period to allow between characters before timing out.

RETURN VALUE

1 if CR or **max** bytes read into **s**.
0 if function times out before reading CR or **max** bytes.

LIBRARY

RS232.LIB

EXAMPLE

```
// echoes NULL terminated character strings
main() {
   int getOk;
   char s[16];
   serAopen(19200);
   loopinit();
   while (1) {
      loophead();
      costate {
         wfd getOk = cof_serAgets (s, 15, 20);
         if (getOk) {
            wfd cof_serAputs(s);
         }
         else {          // timed out: s null terminated,
         }               // but incomplete
      }
   }
   serAclose();
}
```

cof_serXputc

void cof_serXputc(int c); /* where X = A|B|C|D */

DESCRIPTION

This single-user cofunction writes a character to serial port X, yielding to other tasks when the input buffer is locked. This function is non-reentrant.

Starting with Dynamic C version 7.25, the functions **cof_serEputc()** and **cof_serFputc()** may be used with the Rabbit 3000 microprocessor.

PARAMETERS

c Character to write.

LIBRARY

RS232.LIB

EXAMPLE

```
// echoes characters
main() {
   int c;
   serAopen(19200);
   loopinit();
   while (1) {
      loophead();
      wfd c = cof_serAgetc();
      wfd cof_serAputc(c);
   }
   serAclose();
}
```

cof_serXputs

```
void cof_serXputs(char *str); /* where X = A|B|C|D */
```

DESCRIPTION

This single-user cofunction writes a **NULL** terminated string to port X. It yields to other tasks for as long as the input buffer may be locked or whenever the buffer may become full as characters are written. This function is non-reentrant.

Starting with Dynamic C version 7.25, the functions **cof_serEputs()** and **cof_serFputs()** may be used with the Rabbit 3000 microprocessor.

PARAMETERS

str **NULL**-terminated character string to write.

LIBRARY

RS232.LIB

EXAMPLE

```
// writes a NULL-terminated character string, repeatedly
main() {
   const char s[] = "Hello Z-World";
   serAopen(19200);
   loopinit();
   while (1) {
      loophead();
      costate {
         wfd cof_serAputs(s);
      }
   }
   serAclose();
}
```

cof_serXread

```
int cof_serXread(void* data, int length, unsigned long tmout);
   /* where X = A|B|C|D */
```

DESCRIPTION

This single-user cofunction reads **length** characters from port X or until **tmout** milliseconds transpires between characters read. It yields to other tasks for as long as the input buffer is locked or whenever the buffer becomes empty as characters are read. A timeout will never occur if no characters have been read. This function is non-reentrant.

Starting with Dynamic C version 7.25, the functions **cof_serEread()** and **cof_serFread()** may be used with the Rabbit 3000 microprocessor.

PARAMETERS

data	Data structure into which characters are read.
length	The number of characters to read into **data**.
tmout	Millisecond wait period to allow between characters before timing out.

RETURN VALUE

Number of characters read into **data**.

LIBRARY

RS232.LIB

EXAMPLE

```
// echoes a block of characters
main() {
   int n;
   char s[16];
   serAopen(19200);
   loopinit();
   while (1) {
      loophead();
      costate {
         wfd n = cof_serAread(s, 15, 20);
         wfd cof_serAwrite(s, n);
      }
   }
   serAclose();
}
```

cof_serXwrite

```
void cof_serXwrite(void *data, int length);
    /* where X = A|B|C|D */
```

DESCRIPTION

This single-user cofunction writes **length** bytes to port X. It yields to other tasks for as long as the input buffer is locked or whenever the buffer becomes full as characters are written. This function is non-reentrant.

Starting with Dynamic C version 7.25, the functions **cof_serEwrite()** and **cof_serFwrite()** may be used with the Rabbit 3000 microprocessor.

PARAMETERS

data	Data structure to write.
length	Number of bytes in **data** to write.

LIBRARY

RS232.LIB

EXAMPLE

```
// writes a block of characters, repeatedly
main() {
   const char s[] = "Hello Z-World";
   serAopen(19200);
   loopinit();
   while (1) {
      loophead();
      costate {
         wfd cof_serAwrite(s, strlen(s));
      }
   }
   serAclose();
}
```

CoPause

```
void CoPause(CoData *p);
```

DESCRIPTION

Pause execution of a costatement so that it will not run the next time it is encountered unless and until **CoResume(p)** or **CoBegin(p)** are called.

PARAMETERS

p Address of costatement

LIBRARY

COSTATE.LIB

CoReset

```
void CoReset(CoData *p);
```

DESCRIPTION

Initializes a costatement structure so the costatement will not be executed next time it is encountered (unless the costatement is declared to be **always_on**).

PARAMETERS

p Address of costatement

LIBRARY

COSTATE.LIB

CoResume

```
void CoResume(CoData *p);
```

DESCRIPTION

Resume execution of a costatement that has been paused.

PARAMETERS

p	Address of costatement

LIBRARY

COSTATE.LIB

cos

```
float cos(float x);
```

DESCRIPTION

Computes the cosine of real float value **x** (radians).

PARAMETERS

x	Radian value to compute

RETURN VALUE

Cosine of the argument.

LIBRARY

MATH.LIB

SEE ALSO

acos, cosh, sin, tan

cosh

```
float cosh(float x);
```

DESCRIPTION

Computes the hyperbolic cosine of real FLOAT value **x**.

PARAMETERS

x value to compute

RETURN VALUE

Hyperbolic cosine

If |**x**| > 89.8 (approx.), the function returns INF and signals a range error.

LIBRARY

MATH.LIB

SEE ALSO

cos, acos, sin, sinh, tan, tanh

defineErrorHandler

```
void defineErrorHandler(void *errfcn)
```

DESCRIPTION

Sets the BIOS function pointer for runtime errors to the function pointed to by **errfcn**. This user-defined function must be in root memory. Specify **root** at the start of the function definition to ensure this. When a runtime error occurs, the following information is passed to the error handler on the stack:

Stack Position	Stack Contents
SP+0	Return address for **exceptionRet**
SP+2	Error code
SP+4	0x0000 (can be used for additional information)
SP+6	XPC when **exception()** was called (upper byte)
SP+8	Address where **exception()** was called

PARAMETERS

errfcn Pointer to user-defined run-time error handler.

LIBRARY

SYS.LIB

deg

```
float deg(float x);
```

DESCRIPTION

Changes **float** radians **x** to degrees

PARAMETERS

x	Radian value to convert

RETURN VALUE

Angle in degrees (a **float**).

LIBRARY

MATH.LIB

SEE ALSO

rad

DelayMs

```
int DelayMs(long delayms);
```

DESCRIPTION

Millisecond time mechanism for the costatement "waitfor" constructs. The initial call to this function starts the timing. The function returns zero and continues to return zero until the number of milliseconds specified has passed.

PARAMETERS

delayms	The number of milliseconds to wait.

RETURN VALUE

1: The specified number of milliseconds have elapsed
0: The specified number of milliseconds have not elapsed.

LIBRARY

COSTATE.LIB

DelaySec

```
int DelaySec(long delaysec);
```

DESCRIPTION

Second time mechanism for the costatement "waitfor" constructs. The initial call to this function starts the timing. The function returns zero and continues to return zero until the number of seconds specified has passed.

PARAMETERS

delaysec The number of seconds to wait.

RETURN VALUE

1: The specified number of seconds have elapsed.
0: The specified number of seconds have not elapsed.

LIBRARY

COSTATE.LIB

DelayTicks

```
int DelayTicks(unsigned ticks);
```

DESCRIPTION

Tick time mechanism for the costatement "waitfor" constructs. The initial call to this function starts the timing. The function returns zero and continues to return zero until the number of ticks specified has passed.

1 tick = 1/1024 second.

PARAMETERS

ticks The number of ticks to wait.

RETURN VALUE

1: The specified tick delay has elapsed.
0: The specified tick delay has not elapsed.

LIBRARY

COSTATE.LIB

Disable_HW_WDT

```
void Disable_HW_WDT();
```

DESCRIPTION

Disables the hardware watchdog timer on the Rabbit processor. Note that the watchdog will be enabled again just by hitting it. The watchdog is hit by the periodic interrupt, which is on by default. This function is useful for special situations such as low power "sleepy mode."

LIBRARY

SYS.LIB

errlogGetHeaderInfo

```
root char* errlogGetHeaderInfo();
```

DESCRIPTION

Reads the error log header and formats the output.

When running stand alone (not talking to Dynamic C), this function reads the header directly from the log buffer. When in debug mode, this function reads the header from the copy in flash.

When a Dynamic C cold boot takes place, the header in RAM is zeroed out to initialize it, but first its contents are copied to an address in the BIOS code before the BIOS in RAM is copied to flash. This means that on the second cold boot, the data structure in flash will be zeroed out. The configuration of the log buffer may still be read, and the log buffer entries are not affected.

Because the exception mechanism resets the processor by causing a watchdog time-out, the number of watchdog time-outs reported by this functions is the number of actual WDTOs plus the number of exceptions.

RETURN VALUE

A **NULL**-terminated string containing the header information:

```
Status Byte: 0
#Exceptions: 5
Index last exception: 5
#SW Resets: 2
#HW Resets: 2
#WD Timeouts: 5
```

The string will contain "Header checksum invalid" if a checksum error occurs. The meaning of the status byte is as follows:

```
bit 0   - An error has occurred since deployment
bit 1   - The count of SW resets has rolled over.
bit 2   - The count of HW resets has rolled over.
bit 3   - The count of WDTOs has rolled over.
bit 4   - The count of exceptions has rolled over.
bit 5-7 - Not used
```

The index of the last exception is the index from the start of the error log entries. If this index does not equal the total exception count minus one, the error log entries have wrapped around the log buffer.

LIBRARY

ERRORS.LIB

errlogGetNthEntry

```
root int errlogGetNthEntry(int N);
```

DESCRIPTION

Loads **errLogEntry** structure with Nth entry of the error buffer. This must be called before the functions below that format the output.

PARAMETERS

N	Index of entry to load into errLogEntry

RETURN VALUE

0: Success, entry checksum OK.
-1: Failure, entry checksum not OK.

LIBRARY

ERRORS.LIB

errlogFormatEntry

```
root char* errlogFormatEntry();
```

DESCRIPTION

Returns a **NULL**-terminated string containing the basic information contained in **errLogEntry**:

```
Error type=240
Address = 00:16aa
Time: 06/11/2001 20:49:29
```

RETURN VALUE

The **NULL**-terminated string described above.

LIBRARY

ERRORS.LIB

errlogFormatRegDump

```
root char* errlogFormatRegDump();
```

DESCRIPTION

Returns a **NULL**-terminated string containing a register dump using the data in **errLogEntry**:

```
AF=0000,AF'=0000
HL=00f0,HL'=15e3
BC=16ce,BC'=1600
DE=0000,DE'=1731
IX=d3f1,IY =0560
SP=d3eb,XPC=0000
```

RETURN VALUE

The **NULL**-terminated string described above.

LIBRARY

ERRORS.LIB

errlogFormatStackDump

```
root char* errlogFormatStackDump();
```

DESCRIPTION

Returns a **NULL**-terminated string containing a stack dump using the data in **errLogEntry**.

```
Stack Dump:
 0024,04f1,
 d378,c146,
 c400,a108,
 2404,0000,
```

RETURN VALUE

The **NULL**-terminated string describe above.

LIBRARY

ERRORS.LIB

errlogGetMessage

```
root char* errlogGetMessage();
```

DESCRIPTION

Returns a **NULL**-terminated string containing the 8 byte message in **errLogEntry**.

RETURN VALUE

A **NULL**-terminated string.

LIBRARY

ERRORS.LIB

errlogReadHeader

```
root int errlogReadHeader();
```

DESCRIPTION

Reads error log header into the structure **errlogInfo**.

RETURN VALUE

0: Success, entry checksum OK.
-1: Failure, entry checksum not OK.

LIBRARY

ERRORS.LIB

exception

```
int exception(int errCode);
```

DESCRIPTION

This function is called by Rabbit libraries when a runtime error occurs. It puts information relevant to the runtime error on the stack and calls the default runtime error handler pointed to by the **ERROR_EXIT** macro. To define your own error handler, see the **defineErrorHandler()** function.

When the error handler is called, the following information will be on the stack:

Location on Stack	Description
SP+0	Return adress for error handler call
SP+2	Runtime error code
SP+4	(can be used for additional information)
SP+6	XPC when **exception()** was called (upper byte)
SP+8	Address where **exception()** was called from

RETURN VALUE

Runtime error code passed to it.

LIBRARY

ERRORS.LIB

SEE ALSO

defineErrorHandler

exit

```
void exit(int exitcode);
```

DESCRIPTION

Stops the program and returns **exitcode** to Dynamic C. Dynamic C uses values above 128 for run-time errors. When not debugging, **exit** will run an infinite loop, causing a watchdog timeout if the watchdog is enabled.

PARAMETERS

exitcode Error code passed by Dynamic C

LIBRARY

SYS.LIB

exp

```
float exp(float x);
```

DESCRIPTION

Computes the exponential of real **float** value **x**.

PARAMETERS

x Value to compute

RETURN VALUE

Returns the value of e^x.

If **x** > 89.8 (approx.), the function returns INF and signals a range error. If **x** < –89.8 (approx.), the function returns 0 and signals a range error.

LIBRARY

MATH.LIB

SEE ALSO

log, log10, frexp, ldexp, pow, pow10, sqrt

fabs

```
float fabs(float x);
```

DESCRIPTION

Computes the float absolute value of **float x**.

PARAMETERS

x Value to compute

RETURN VALUE

x, if **x** >= 0,
else **-x**.

LIBRARY

MATH.LIB

SEE ALSO

abs

fclose

```
void fclose(File* f);
```

DESCRIPTION

Closes a file.

PARAMETERS

f	The pointer to the file to close.

LIBRARY

FILESYSTEM.LIB

fcreate (FS1)

```
int fcreate(File* f, FileNumber fnum);
```

DESCRIPTION

Creates a file. Before calling this function, a variable of type **File** must be defined in the application program.

```
File file;
fcreate (&file, 1);
```

PARAMETERS

f	The pointer to the created file.
fnum	This is a user-defined number in the range of 1 to127 inclusive. Each file in the flash file system is assigned a unique number in this range.

RETURN VALUE

0: Success.
1: Failure.

LIBRARY

FILESYSTEM.LIB

fcreate (FS2)

```
int fcreate(File* f, FileNumber name);
```

DESCRIPTION

Create a new file with the given "file name" which is composed of two parts: the low byte is the actual file number (1 to 255 inclusive), and the high byte contains an extent number (1 to **_fs.num_lx**) on which to place the file metadata. The extent specified by **fs_set_lx()** is always used to determine the actual data extent. If the high byte contains 0, then the default metadata extent specified by **fs_set_lx()** is used. The file descriptor is filled in if successful. The file will be opened for writing, so a further call to **fopen_wr()** is not necessary.

The number of files which may be created is limited by the lower of **FS_MAX_FILES** and 255. This limit applies to the entire filesystem (all logical extents).

Once a file is created, its data and metadata extent numbers are fixed for the life of the file, i.e. until is is deleted.

When created, no space is allocated in the file system until the first write occurs for the file. Thus, if the system power is cycled after creation but before the first byte is written, the file will be effectively deleted. The first write to a file causes one sector to be allocated for the metadata.

Before calling this function, a variable of type **File** must be defined in the application program. (The **sizeof()** function will return the number of bytes used for the **File** data structure.)

```
File file;
fcreate (&file, 1);
```

PARAMETERS

f	Pointer to the file descriptor to fill in.
name	File number including optional metadata extent number.

RETURN VALUE

0: Success.
!0: Failure.

ERRNO VALUES

EINVAL - Zero file number requested, or invalid extent number.
EEXIST - File with given number already exists.
ENFILE - No space is available in the "existing fileable". If this error occurs, increase the definition of **FS_MAX_FILES**, which is a **#define** constant which should be declared before **#use "fs2.lib"**.

LIBRARY

fs2.LIB

SEE ALSO

fcreate_unused (FS2), fs_set_lx (FS2), fdelete (FS2)

fcreate_unused (FS1)

```
FileNumber fcreate_unused(File* f);
```

DESCRIPTION

Searches for the first unused file number in the range 1 through 127, and creates a file with that number.

PARAMETERS

f	The pointer to the created file.

RETURN VALUE

The **FileNumber** (1-127) of the new file if success.

LIBRARY

FILESYSTEM.LIB

SEE ALSO

fcreate (FS1)

fcreate_unused (FS2)

```
FileNumber fcreate_unused(File* f);
```

DESCRIPTION

Create a new file and return the "file name" which is a number between 1 and 255. The new file will be created on the current default extent(s) as specified by **fs_set_lx()**. Other behavior is the same as **fcreate()**.

PARAMETERS

f	Pointer to file descriptor to fill in.

RETURN VALUE

>0: Success, the **FileNumber** (1-255) of the new file.
0: Failure.

ERRNO VALUE

ENFILE - No unused file number available.

LIBRARY

fs2.LIB

SEE ALSO

fcreate (FS2), fs_set_lx (FS2), fdelete (FS2)

fdelete (FS1)

```
int fdelete(FileNumber fnum);
```

DESCRIPTION

Deletes a file.

PARAMETERS

fnum	A number in the range 1 to 127 inclusive that identifies the file in the flash file system.

RETURN VALUE

0: Success.
1: Failure.

LIBRARY

FILESYSTEM.LIB

fdelete (FS2)

```
int fdelete(FileNumber name);
```

DESCRIPTION

Delete the file with the given number. The specified file must not be open. The file number (i.e. **name**) is composed of two parts: the low byte contains the actual file number, and the high byte (if not zero) contains the metadata extent number of the file.

PARAMETERS

name File number (1 to 255 inclusive).

RETURN VALUE

0: Success.
!0: Failure.

LIBRARY

fs2.LIB

ERRNO VALUES

ENOENT - File does not exist, or metadata extent number does not match an existing file.
EBUSY - File is open.
EIO - I/O error when releasing blocks occupied by this file.

SEE ALSO

fcreate (FS2)

fflush (FS2)

```
int fflush(File * f);
```

DESCRIPTION

Flush any buffers, associated with the given file, retained in RAM to the underlying hardware device. This ensures that the file is completely written to the filesystem. The file system does not currently perform any buffering, however future revisions of this library may introduce buffering to improve performance.

PARAMETERS

f	Pointer to open file descriptor.

RETURN VALUE

0: Success.
!0: Failure.

ERRNO VALUES

EBADFD - file invalid or not open.
EIO - I/O error.

LIBRARY

fs2.LIB

SEE ALSO

fs_sync (FS2)

fftcplx

```
void fftcplx(int *x, int N, int *blockexp)
```

DESCRIPTION

Computes the complex DFT of the **N**-point complex sequence contained in the array **x** and returns the complex result in *x*. **N** must be a power of 2 and lie between 4 and 1024. An invalid **N** causes a RANGE exception. The **N**-point complex sequence in array **x** is replaced with its **N**-point complex spectrum. The value of **blockexp** is increased by 1 each time array **x** has to be scaled, to avoid arithmetic overflow.

PARAMETERS

x	Pointer to **N**-element array of complex fractions.
N	Number of complex elements in array **x**.
blockexp	Pointer to integer block exponent.

LIBRARY

FFT.LIB

SEE ALSO

fftcplxinv, fftreal, fftrealinv, hanncplx, hannreal, powerspectrum

fftcplxinv

```
void fftcplxinv(int *x, int N, int *blockexp)
```

DESCRIPTION

Computes the inverse complex DFT of the **N**-point complex spectrum contained in the array **x** and returns the complex result in **x**. **N** must be a power of 2 and lie between 4 and 1024. An invalid **N** causes a RANGE exception. The value of **blockexp** is increased by 1 each time array **x** has to be scaled, to avoid arithmetic overflow. The value of **blockexp** is also *decreased* by $\log_2$**N** to include the 1/**N** factor in the definition of the inverse DFT

PARAMETERS

x	Pointer to **N**-element array of complex fractions.
N	Number of complex elements in array **x**.
blockexp	Pointer to integer block exponent.

LIBRARY

FFT.LIB

SEE ALSO

fftcplx, fftreal, fftrealinv, hanncplx, hannreal, powerspectrum

fftreal

```
void fftreal(int *x, int N, int *blockexp)
```

DESCRIPTION

Computes the **N**-point, positive-frequency complex spectrum of the 2**N**-point real sequence in array **x**. The 2**N**-point real sequence in array **x** is replaced with its **N**-point positive-frequency complex spectrum. The value of **blockexp** is increased by 1 each time array **x** has to be scaled, to avoid arithmetic overflow.

The imaginary part of the *X*[0] term (stored in *x*[1]) is set to the real part of the *fmax* term.

The 2**N**-point real sequence is stored in natural order. The zeroth element of the sequence is stored in **x[0]**, the first element in **x[1]**, and the *k*th element in *x*[*k*].

N must be a power of 2 and lie between 4 and 1024. An invalid **N** causes a RANGE exception.

PARAMETERS

x	Pointer to 2**N**-point sequence of real fractions.
N	Number of complex elements in output spectrum
blockexp	Pointer to integer block exponent.

LIBRARY

FFT.LIB

SEE ALSO

fftcplx, fftcplxinv, fftrealinv, hanncplx, hannreal, powerspectrum

fftrealinv

```
void fftrealinv(int *x, int N, int *blockexp)
```

DESCRIPTION

Computes the *2***N**-point real sequence corresponding to the **N**-point, positive-frequency complex spectrum in array **x**. The **N**-point, positive-frequency spectrum contained in array **x** is replaced with its corresponding 2**N**-point real sequence. The value of **blockexp** is increased by 1 each time array **x** has to be scaled, to avoid arithmetic overflow. The value of **blockexp** is also *decreased* by $\log_2$**N** to include the 1/**N** factor in the definition of the inverse DFT.

The function expects to find the real part of the *fmax* term in the imaginary part of the zero-frequency **X[0]** term (stored **x[1]**).

The 2**N**-point real sequence is stored in natural order. The zeroth element of the sequence is stored in **x[0]**, the first element in **x[1]**, and the *k*th element in **x[k]**.

N must be a power of 2 and lie between 4 and 1024. An invalid **N** causes a RANGE exception.

PARAMETERS

x	Pointer to **N**-element array of complex fractions.
N	Number of complex elements in array **x**.
blockexp	Pointer to integer block exponent.

LIBRARY

FFT.LIB

SEE ALSO

fftcplx, fftcplxinv, fftreal, hanncplx, hannreal, powerspectrum

flash_erasechip

```
void flash_erasechip(FlashDescriptor* fd);
```

DESCRIPTION

Erases an entire Flash Memory chip.

NOTE: **fd** must have already been initialized with **flash_init** before calling this function. See **flash_init** description for further restrictions.

PARAMETERS

fd	Pointer to flash descriptor of the chip to erase.

LIBRARY

FLASH.LIB

SEE ALSO

flash_erasesector, flash_gettype, flash_init, flash_read, flash_readsector, flash_sector2xwindow, flash_writesector

flash_erasesector

```
int flash_erasesector(FlashDescriptor* fd, word which);
```

DESCRIPTION

Erases a sector of a Flash Memory chip.

NOTE: **fd** must have already been initialized with **flash_init** before calling this function. See **flash_init** description for further restrictions.

PARAMETERS

fd	Pointer to flash descriptor of the chip to erase a sector of.
which	The sector to erase.

RETURN VALUE

0: Success.

LIBRARY

FLASH.LIB

SEE ALSO

flash_erasechip, flash_gettype, flash_init, flash_read, flash_readsector, flash_sector2xwindow, flash_writesector

flash_gettype

```
int flash_gettype(FlashDescriptor* fd);
```

DESCRIPTION

Returns the 16-bit Flash Memory type of the Flash Memory.

NOTE: **fd** must have already been initialized with **flash_init** before calling this function. See **flash_init** description for further restrictions.

PARAMETERS

fd	The **FlashDescriptor** of the memory to query.

RETURN VALUE

The integer representing the type of the Flash Memory.

LIBRARY

FLASH.LIB

SEE ALSO

flash_erasechip, flash_erasesector, flash_init, flash_read, flash_readsector, flash_sector2xwindow, flash_writesector

flash_init

```
int flash_init(FlashDescriptor* fd, int mb3cr);
```

DESCRIPTION

Initializes an internal data structure of type **FlashDescriptor** with information about the Flash Memory chip. The Memory Interface Unit bank register (MB3CR) will be assigned the value of **mb3cr** whenever a function accesses the Flash Memory referenced by **fd**. See the Rabbit 2000 Users Manual for the correct chip select and wait state settings.

NOTE: Improper use of this function can cause your program to be overwritten or operate incorrectly. This and the other Flash Memory access functions should not be used on the same Flash Memory that your program resides on, nor should they be used on the same region of a second Flash Memory where a file system resides.

Use **WriteFlash()** to write to the primary Flash Memory.

PARAMETERS

fd	This is a pointer to an internal data structure that holds information about a Flash Memory chip.
mb3cr	This is the value to set MB3CR to whenever the Flash Memory is accessed. 0xc2 (i.e., CS2, /OE0, /WE0, 0 WS) is a typical setting for the second Flash Memory on the TCP/IP Dev Kit, the Intellicom, the Advanced Ethernet Core, and the RabbitLink.

RETURN VALUE

0: Success.
1: Invalid Flash Memory type.
-1: Attempt made to initialize primary Flash Memory.

LIBRARY

FLASH.LIB

SEE ALSO

flash_erasechip, flash_erasesector, flash_gettype, flash_read, flash_readsector, flash_sector2xwindow, flash_writesector

flash_read

```
int flash_read(FlashDescriptor* fd, word sector, word offset,
   unsigned long buffer, word length);
```

DESCRIPTION

Reads data from the Flash Memory and stores it in **buffer**.

NOTE: **fd** must have already been initialized with **flash_init** before calling this function. See the **flash_init** description for further restrictions.

PARAMETERS

fd	The **FlashDescriptor** of the Flash Memory to read from.
sector	The sector of the Flash Memory to read from.
offset	The displacement, in bytes, from the beginning of the sector to start reading at.
buffer	The physical address of the destination buffer. TIP: A logical address can be changed to a physical with the function **paddr**.
length	The number of bytes to read.

RETURN VALUE

0: Success.

LIBRARY

FLASH.LIB

SEE ALSO

flash_erasechip, flash_erasesector, flash_gettype, flash_init, flash_readsector, flash_sector2xwindow, flash_writesector, paddr

flash_readsector

```
int flash_readsector(FlashDescriptor* fd, word sector, unsigned
   long buffer);
```

DESCRIPTION

Reads the contents of an entire sector of Flash Memory into a buffer.

NOTE: **fd** must have already been initialized with **flash_init** before calling this function. See **flash_init** description for further restrictions.

PARAMETERS

fd	The **FlashDescriptor** of the Flash Memory to read from.
sector	The source sector to read.
buffer	The physical address of the destination buffer. TIP: A logical address can be changed to a physical with the function **paddr**.

RETURN VALUE

0: Success.

LIBRARY

FLASH.LIB

SEE ALSO

flash_erasechip, flash_erasesector, flash_gettype, flash_init, flash_read, flash_sector2xwindow, flash_writesector

flash_sector2xwindow

```
void* flash_sector2xwindow(FlashDescriptor* fd, word sector);
```

DESCRIPTION

This function sets the MB3CR and XPC value so the requested sector falls within the XPC window. The MB3CR is the Memory Interface Unit bank register. XPC is one of four Memory Management Unit registers. See **flash_init** description for restrictions.

PARAMETERS

fd	The **FlashDescriptor** of the Flash Memory.
sector	The sector to set the XPC window to.

RETURN VALUE

The logical offset of the sector.

LIBRARY

FLASH.LIB

SEE ALSO

flash_erasechip, flash_erasesector, flash_gettype, flash_init, flash_read, flash_readsector, flash_writesector

flash_writesector

```
int flash_writesector(FlashDescriptor* fd, word sector,
   unsigned long buffer);
```

DESCRIPTION

Writes the contents of **buffer** to **sector** on the Flash Memory referenced by **fd**. NOTE: **fd** must have already been initialized with **flash_init** before calling this function. See **flash_init** description for further restrictions.

PARAMETERS

fd	The **FlashDescriptor** of the Flash Memory to write to.
sector	The destination sector.
buffer	The physical address of the source. TIP: A logical address can be changed to a physical address with the function **paddr**

RETURN VALUE

0: Success.

LIBRARY

FLASH.LIB

SEE ALSO

flash_erasechip, flash_erasesector, flash_gettype, flash_init, flash_read, flash_readsector, flash_sector2xwindow

floor

```
float floor(float x);
```

DESCRIPTION

Computes the largest integer less than or equal to the given number.

PARAMETERS

x Value to round down

RETURN VALUE

Rounded down value.

LIBRARY

MATH.LIB

SEE ALSO

ceil, fmod

fmod

```
float fmod(float x, float y);
```

DESCRIPTION

Calculates modulo math.

PARAMETERS

x Dividend

y Divisor

RETURN VALUE

Returns the remainder of x/y. The remaining part of **x** after all multiples of **y** have been removed. For example, if **x** is 22.7 and **y** is 10.3, the integral division result is 2. Then the remainder is: $22.7 - 2 \times 10.3 = 2.1$.

LIBRARY

MATH.LIB

SEE ALSO

ceil, floor

fopen_rd (FS1)

```
int fopen_rd(File* f, FileNumber fnum);
```

DESCRIPTION

Opens a file for reading.

PARAMETERS

f	A pointer to the file to read.
fnum	A number in the range 1 to 127 inclusive that identifies the file in the flash file system.

RETURN VALUE

0: Success.
1: Failure.

LIBRARY

FILESYSTEM.LIB

fopen_rd (FS2)

```
int fopen_rd(File* f, FileNumber name);
```

DESCRIPTION

Open file for reading only. See **fopen_wr()** for a more detailed description.

PARAMETERS

f	Pointer to file descriptor (uninitialized).
name	File number (1 to 255 inclusive).

RETURN VALUE

0: Success.
!0: Failure.

ERRNO VALUES

ENOENT - File does not exist, or metadata extent number does not match an existing file.

LIBRARY

fs2.lib

SEE ALSO

fclose, fopen_wr (FS2)

fopen_wr (FS1)

```
int fopen_wr(File* f, FileNumber fnum);
```

DESCRIPTION

Opens a file for writing.

PARAMETERS

f	A pointer to the file to write.
fnum	A number in the range 1 to 127 inclusive that identifies the file in the flash file system.

RETURN VALUE

0: Success.
1: Failure.

LIBRARY

FILESYSTEM.LIB

fopen_wr (FS2)

```
int fopen_wr(File* f, FileNumber name);
```

DESCRIPTION

Open file for read or write. The given file number is composed of two parts: the low byte contains the file number (1 to 255 inclusive) and the high byte, if not zero, contains the metadata extent number. If the extent number is zero, it defaults to the correct metadata extent - this is for the purpose of validating an expected extent number. Most applications should just pass the file number with zero high byte.

A file may be opened multiple times, with a different file descriptor pointer for each call, which allows the file to be read or written at more than one position at a time. A reference count for the actual file is maintained, so that the file can only be deleted when all file descriptors referring to this file are closed.

`fopen_wr()` or **`fopen_rd()`** must be called before any other function from this library is called that requires a **`File`** pointer. The "current position" is set to zero i.e. the start of the file.

When a file is created, it is automatically opened for writing thus a subsequent call to **`fopen_wr()`** is redundant.

PARAMETERS

`f`	Pointer to file descriptor (uninitialized).
`name`	File number (1 to 255 inclusive).

RETURN VALUE

`0`: Success.
`!0`: Failure.

ERRNO VALUES

`ENOENT` - File does not exist, or metadata extent number does not match an existing file.

LIBRARY

fs2.lib

SEE ALSO

`fclose, fopen_rd (FS2)`

forceSoftReset

```
void forceSoftReset();
```

DESCRIPTION

Forces the board into a software reset by jumping to the start of the BIOS.

LIBRARY

SYS.LIB

fread (FS1)

```
int fread(File* f, char* buf, int len);
```

DESCRIPTION

Reads **len** bytes from a file pointed to by **f**, starting at the current offset into the file, into buffer. Data is read into buffer pointed to by **buf**.

PARAMETERS

f	A pointer to the file to read from
buf	A pointer to the destination buffer.
len	Number of bytes to copy.

RETURN VALUE

Number of bytes read.

LIBRARY

FILESYSTEM.LIB

fread (FS2)

```
int fread(File* f, void* buf, int len);
```

DESCRIPTION

Read data from the "current position" of the given file. When the file is opened, the current position is 0, i.e. at the start of file. Subsequent reads or writes advance the position by the number of bytes read/written **fseek()** can also be used to position the read point.

If the application permits, it is much more efficient to read multiple data bytes rather than reading one-by-one.

PARAMETERS

f	Pointer to file descriptor (initialized by **fopen_rd()**, **fopen_wr()** or **fcreate()**).
buf	Data buffer located in root data memory or stack. This must be dimensioned with at least len bytes.
len	Length of data to read (0 to 32767 inclusive).

RETURN VALUE

len: Success.

<len : Partial success. Returns amount successfully read. **errno** gives further details (probably 0 meaning that end-of-file was encountered).

0: Failure, or **len** was zero.

LIBRARY

fs2.LIB

ERRNO VALUES

EBADFD - File descriptor not opened.
EINVAL - **len** less than zero.
0 - Success, but **len** was zero or EOF was reached prior to reading **len** bytes.
EIO - I/O error.

SEE ALSO

fseek (FS2), fwrite (FS2)

frexp

```
float frexp(float x, int *n);
```

DESCRIPTION

Splits **x** into a fraction and exponent, f*(2**n)

PARAMETERS

x	Number to split
n	An integer

RETURN VALUE

The function returns the exponent in the integer ***n** and the fraction between 0.5, inclusive and 1.0.

LIBRARY

MATH.LIB

SEE ALSO

exp, ldexp

fs_format (FS1)

```
int fs_format(long reserveblocks, int num_blocks, unsigned long
   wearlevel);
```

DESCRIPTION

Initializes the internal data structures and file system. All blocks in the file system are erased.

PARAMETERS

reserveblocks	Starting address of the flash file system. When **FS_FLASH** is defined this value should be 0 or a multiple of the block size. When **FS_RAM** is defined this parameter is ignored.
num_blocks	The number of blocks to allocate for the file system. With a default block size of 4096 bytes and a 256K Flash Memory, this value might be 64.
wearlevel	This value should be 1 on a new Flash Memory, and some higher value on an unformatted used Flash Memory. If you are reformatting a Flash Memory you can set **wearlevel** to 0 to keep the old wear leveling.

RETURN VALUE

0: Success.
1: Failure.

LIBRARY

FILESYSTEM.LIB

EXAMPLE

This program can be found in **samples/filesystem/format.c**.

```
#define FS_FLASH
#use "filesystem.lib"
#define RESERVE   0
#define BLOCKS   64
#define WEAR      1

main() {
   if(fs_format(RESERVE,BLOCKS,WEAR)) {
      printf("error formating flash\n");
   } else {
      printf("flash successfully formatted\n");
   }
}
```

fs_format (FS2)

```
int fs_format(long reserveblocks, int num_blocks, unsigned
   wearlevel)
```

DESCRIPTION

Format all extents of the file system. This must be called after calling **fs_init()**. Only extents that are not defined as reserved are formatted. All files are deleted.

PARAMETERS

reserveblocks	Must be zero. Retained for backward compatibility.
num_blocks	Ignored (backward compatibility).
wearlevel	Initial wearlevel value. This should be 1 if you have a new flash, and some larger number if the flash is used. If you are reformatting a flash, you can use 0 to use the old flash wear levels.

RETURN VALUE

0: Success.
!0: Failure.

ERRNO VALUES

EINVAL - the **reserveblocks** parameter was non-zero.
EBUSY - one or more files were open.
EIO - I/O error during format. If this occurs, retry the format operation If it fails again, there is probably a hardware error.

SEE ALSO

fs_init (FS2), lx_format

fs_init (FS1)

```
int fs_init(long reserveblocks, int num_blocks);
```

DESCRIPTION

Initialize the internal data structures for an existing file system. Blocks that are used by a file are preserved and checked for data integrity.

PARAMETERS

reserveblocks	Starting address of the flash file system. When **FS_FLASH** is defined this value should be 0 or a multiple of the block size. When **FS_RAM** is defined this parameter is ignored.
num_blocks	The number of blocks that the file system contains. By default the block size is 4096 bytes.

RETURN VALUE

0:Success.
1: Failure.

LIBRARY

FILESYSTEM.LIB

fs_init (FS2)

```
int fs_init(long reserveblocks, int num_blocks);
```

DESCRIPTION

Initialize the filesystem. The static structure **_fs** contains information that defines the number and parameters associated with each extent or "partition". This function must be called before any of the other functions in this library, except for **fs_setup()**, **fs_get_*_lx()** and **fs_get_lx_size()**.

Pre-main initialization will create up to 3 devices:

- The second flash device (if available on the board)
- Battery-backed SRAM (if **FS2_RAM_RESERVE** defined)
- The first (program) flash (if both **XMEM_RESERVE_SIZE** and **FS2_USE_PROGRAM_FLASH** defined)

The LX numbers of the default devices can be obtained using the **fs_get_flash_lx()**, **fs_get_ram_lx()** and **fs_get_other_lx()** calls.

If none of these devices can be set up successfully, **fs_init()** will return **ENOSPC** when called.

This function performs complete consistency checks and, if necessary, fixups for each LX. It may take up to several seconds to run. It should only be called once at application initialization time.

PARAMETERS

reserveblocks	Must be zero. Retained for backward compatibility.
num_blocks	Ignored (backward compatibility).

RETURN VALUE

0: Success.
!0: Failure.

ERRNO VALUES

EINVAL - the reserveblocks parameter was non-zero.
EIO - I/O error. This indicates a hardware problem.
ENOMEM - Insufficient memory for required buffers.
ENOSPC - No valid extents obtained e.g. there is no recognized flash or RAM memory device available.

LIBRARY

fs2.lib

SEE ALSO

fs_setup (FS2), fs_get_flash_lx (FS2)

fs_reserve_blocks (FS1)

```
int fs_reserve_blocks(int blocks);
```

DESCRIPTION

Sets up a number of blocks that are guaranteed to be available for privileged files. A privileged file has an identifying number in the range 128 through 143. This function is not needed in most cases. If it is used, it should be called immediately after **fs_init** or **fs_format**.

PARAMETERS

blocks Number of blocks to reserve.

RETURN VALUE

0: Success.
1: Failure.

LIBRARY

FILESYSTEM.LIB

fsck (FS1)

```
int fsck(int flash);
```

DESCRIPTION

Check the filesystem for errors

PARAMETERS

flash A bitmask indicating which checks to **NOT** perform. The following checks are available:

FSCK_HEADERS - Block headers.
FSCK_CHECKSUMS - Data checksums.
FSCK_VERSION - Block versions, from a failed write.

RETURN VALUE

0: Success.
!0: Failure, this is a bitmask indicating which checks failed.

LIBRARY

FILESYSTEM.LIB

fseek (FS1)

```
int fseek(File* f, long to, char whence);
```

DESCRIPTION

Places the read pointer at a desired location in the file.

PARAMETERS

f	A pointer to the file to seek into.
to	The number of bytes to move the read pointer. This can be a positive or negative number.
whence	The location in the file to offset from. This is one of the following constants. **SEEK_SET** - Seek from the beginning of the file. **SEEK_CUR** - Seek from the current read position in the file. **SEEK_END** - Seek from the end of the file.

EXAMPLE

To seek to 10 bytes from the end of the file **f**, use **fseek(f, -10, SEEK_END);**.
To rewind the file **f** by 5 bytes, use **fseek(f, -5, SEEK_CUR);**.

RETURN VALUE

0: Success.
1: Failure.

LIBRARY

FILESYSTEM.LIB

fseek (FS2)

```
int fseek(File * f, long where, char whence);
```

DESCRIPTION

Set the current read/write position of the file. Bytes in a file are sequentially numbered starting at zero. If the current position is zero, then the first byte of the file will be read or written. If the position equals the file length, then no data can be read, but any write will append data to the file.

fseek() allows the position to be set relative to the start or end of the file, or relative to its current position.

In the special case of **SEEK_RAW**, an unspecified number of bytes beyond the "known" end-of-file may be readable. The actual amount depends on the amount of space left in the last internal block of the file. This mode only applies to reading, and is provided for the purpose of data recovery in the case that the application knows more about the file structure than the filesystem.

PARAMETERS

f	Pointer to file descriptor (initialized by **fopen_rd()**, **fopen_wr()** or **fcreate()**).
where	New position, or offset.
whence	One of the following values: **SEEK_SET**: 'where' (non-negative only) is relative to start of file. **SEEK_CUR**: 'where' (positive or negative) is relative to the current position. **SEEK_END**: 'where' (non-positive only) is relative to the end of the file. **SEEK_RAW**: Similar to **SEEK_END**, except the file descriptor is set in a special mode which allows reading beyond the end of the file.

RETURN VALUE

0: Success.

!0: The computed position was outside of the current file contents, and has been adjusted to the nearest valid position.

ERRNO VALUES

None.

LIBRARY

fs2.lib

SEE ALSO

ftell (FS2), fread (FS2), fwrite (FS2)

fs_get_flash_lx (FS2)

```
FSLXnum fs_get_flash_lx(void);
```

DESCRIPTION

Returns the logical extent number of the preferred flash device. This is the second flash if one is available on your hardware, otherwise it is the reserved area in your program flash. In order for the program flash to be available for use by the file system, you must define two constants: the first constant is **XMEM_RESERVE_SIZE** near the top of **BIOS\RABBITBIOS.C**. This value is set to the amount of program flash to reserve (in bytes). This is required by the BIOS. The second constant is set in your code before **#use "fs2.lib"**. **FS2_USE_PROGRAM_FLASH** must be defined to the number of KB (1024 bytes) that will actually be used by the file system. If this is set to a larger value than the actual amount of reserved space, then only the actual amount will be used.

The sample program **SAMPLES\FILESYSTEM\FS2INFO.C** demonstrates use of this function.

This function may be called before calling **fs_init()**.

RETURN VALUE

0: There is no flash file system available.
!0: Logical extent number of the preferred flash.

LIBRARY

FS2.lib

SEE ALSO

fs_get_ram_lx (FS2), fs_get_other_lx (FS2)

fs_get_lx (FS2)

```
FSLXnum fs_get_lx(int meta);
```

DESCRIPTION

Return the current extent (LX) number for file creation. Each file has two parts: the main bulk of data, and the metadata which is a relatively small, fixed, amount of data used to journal changes to the file. Both data and metadata can reside on the same extent, or they may be separated.

PARAMETERS

meta	**1**: return logical extent number for metadata. **0**: return logical extent number for data.

RETURN VALUE

Logical extent number.

LIBRARY

FS2.lib

SEE ALSO

fcreate (FS2), fs_set_lx (FS2)

fs_get_lx_size (FS2)

```
long fs_get_lx_size(FSLXnum lxn, int all, word ls_shift);
```

DESCRIPTION

Returns the size of the specified logical extent, in bytes. This information is useful when initially partitioning an LX, or when estimating the capacity of an LX for user data. **all** is a flag which indicates whether to return the total data capacity (as if all current files were deleted) or whether to return just the available data capacity. The return value accounts for the packing efficiency which will be less than 100% because of the bookkeeping overhead. It does not account for the free space required when any updates are performed; however this free space may be shared by all files on the LX. It also does not account for the space required for file metadata. You can account for this by adding one logical sector for each file to be created on this LX. You can also specify that the metadata be stored on a different LX by use of **fs_set_lx()**.

This function may be called either before or after **fs_init()**. If called before, then the **ls_shift** parameter must be set to the value to be used in **fs_setup()**, since the LS size is not known at this point. **ls_shift** can also be passed as zero, in which case the default size will be assumed. **all** must be non-zero if called before **fs_init()**, since the number of files in use is not yet known.

PARAMETERS

lxn	Logical extent number to query.
all	Boolean: 0 for current free capacity only, 1 for total. Must use 1 if calling before **fs_init()**.
ls_shift	Logical sector shift i.e. log base 2 of LS size (6 to 13); may be zero to use default.

RETURN VALUE

0: The specified LX does not exist.
!0: Capacity of the LX in bytes.

LIBRARY

FS2.lib

fs_get_other_lx (FS2)

```
FSLXnum fs_get_other_lx(void);
```

DESCRIPTION

Returns the logical extent number of the non-preferred flash device. If it exists, this is usually the program flash. See the description under **fs_get_flash_lx()** for details about setting up the program flash for use by the filesystem.

The sample program **SAMPLES\FILESYSTEM\FS2INFO.C** demonstrates use of this function.

This function may be called before calling **fs_init()**.

RETURN VALUE

0: There is no other flash filesystem available.
!0: Logical extent number of the non-preferred flash.

LIBRARY

FS2.LIB

SEE ALSO

fs_get_ram_lx (FS2), fs_get_flash_lx (FS2)

fs_get_ram_lx (FS2)

```
FSLXnum fs_get_ram_lx(void);
```

DESCRIPTION

Return the logical extent number of the RAM file system device. This is only available if you have defined **FS2_RAM_RESERVE** to a non-zero number of bytes in the BIOS.

A RAM filesystem is only really useful if you have battery-backed SRAM on the board. You can still use a RAM file system on volatile RAM, but of course files will not persist over power cycles and you should explicitly format the RAM filesystem at power-up.

The sample program **SAMPLES\FILESYSTEM\FS2INFO.C** demonstrates use of this function.

This function may be called before calling **fs_init()**.

RETURN VALUE

0: There is no RAM filesystem available.
!0: Logical extent number of the RAM device.

LIBRARY

FS2.LIB

SEE ALSO

fs_get_flash_lx (FS2), fs_get_other_lx (FS2)

fs_set_lx (FS2)

```
int fs_set_lx(FSLXnum meta, FSLXnum data);
```

DESCRIPTION

Sets the default logical extent (LX) numbers for file creation. Each file has two parts: the main bulk of data, and the metadata which is a relatively small, fixed amount of data used to journal changes to the file. Both data and metadata can reside on the same extent, or they may be separated. The metadata, no matter where it is located, consumes one sector.

The file creation functions allow the metadata extent to be explicitly specified (in the high byte of the file number), however it is usually easier to call **fs_set_lx()** to set appropriate defaults. Calling **fs_set_lx()** is the only way to specify the data extent.

If **fs_set_lx()** is never called, both data and metadata will default to the first non-reserved extent number.

PARAMETERS

meta	Extent number for metadata.
data	Extent number for data.

RETURN VALUE

0: Success.
!0: Error, e.g. non-existent LX number.

ERRNO VALUES

ENODEV - no such extent number, or extent is reserved.

LIBRARY

FS2.LIB

SEE ALSO

fcreate (FS2)

fs_setup (FS2)

```
FSLXnum  fs_setup(FSLXnum lxn, word ls_shift, int reserve_it,
   void * rfu, int partition_it, word part, word part_ls_shift,
   int part_reserve, void * part_rfu);
```

DESCRIPTION

To modify or add to the default extents, this function must be called before calling **fs_init()**. If called after **fs_init()**, the filesystem will be corrupted.

fs_setup() runs in one of two basic modes, determined by the **partition_it** parameter. If **partition_it** is non-zero, then the specified extent (**lxn**, which must exist), is split into two extents according to the given proportions. If **partition_it** is zero, then the specified extent must not exist; it is created. This use is beyond the scope of this note, since it involves filesystem internals. The parition-ing usage is described here.

partition_it may be **FS_MODIFY_EXTENT** in which case the base extent, **lxn**, is modified to use the specified **ls_shift** and **reserve_it** parameters (the other parameters are ignored).

partition_it may be set to **FS_PARTITION_FRACTION** (other values re-served). This causes extent number **lxn** to be split. The first half is still referred to as extent **lxn**, and the other half is assigned a new extent number, which is returned.

The base extent number may itself have been previously partitioned, or it should be 1 for the 2nd flash device, or possibly 2 for the NVRAM device.

PARAMETERS

lxn	Base extent number to partition or modify.
ls_shift	New logical sector size to assign to base partition, or zero to not alter it. This is expressed as the log base 2 of the desired size, and must be a number between 6 and 13 in-clusive.
reserve_it	**TRUE** if base partition is to be marked reserved.
rfu	A pointer reserved for future use. Pass as **NULL**.
partition_it	Must be set to **FS_PARTITION_FRACTION** or **FS_MODIFY_EXTENT**. The following parameters are ignored if this parameter is not **FS_PARTITION_FRACTION**.

part	The fraction of the existing base extent to assign to the new extent. This number is expressed as a fixed-point binary number with the binary point to the left of the MSB e.g. 0x3000 assigns 3/16 of the base extent to the new partition, updating the base extent to 13/16 of its original size. The nearest whole number of physical sectors is used for each extent.
part_ls_shift	Logical sector size to assign to the new extent, or zero to use the same LS size as the base extent. Expressed in same units as parameter 2.
part_reserve	**TRUE** if the new extent is to be reserved.
part_rfu	A pointer reserved for future use. Pass as **NULL**.

RETURN VALUE

0: Failure, extent could not be partitioned.
!0: Success, number of the new extent, or same as **lxd** for existing extent modification.

ERRNO VALUES

ENOSPC - one or other half would contain an unusably small number of logical sectors, or the extent table is full. In the latter case, **#define FS_MAX_LX** to a larger value.

EINVAL - **partition_it** set to an invalid value, or other parameter invalid.

ENODEV - specified base extent number not defined.

LIBRARY

FS2.LIB

SEE ALSO

fs_init (FS2)

fs_sync (FS2)

```
int fs_sync(void);
```

DESCRIPTION

Flush any buffers retained in RAM to the underlying hardware device. The file system does not currently perform any buffering, however future revisions of this library may introduce buffering to improve performance. This function is similar to **fflush()**, except that the entire file system is synchronized instead of the data for just one file. Use **fs_sync()** in preference to **fflush()** if there is only one extent in the filesystem.

RETURN VALUE

0: Success.
!0: Failure.

ERRNO VALUES

EIO - I/O error.

LIBRARY

FS2.LIB

SEE ALSO

fflush (FS2)

ftell (FS1)

```
long ftell(File* f);
```

DESCRIPTION

Gets the offset from the beginning of a file that the read pointer is currently at.

TIP: **ftell()** can be used with **fseek()** to find the length of a file.

```
fseek(f, 0, SEEK_END); /* seek to the end of the file */
FileLength = ftell(f);  /* find the length of the file */
```

PARAMETERS

f	A pointer to the file to query.

RETURN VALUE

The offset in bytes of the read pointer from the beginning of the file: Success.
-1: Failure.

LIBRARY

FILESYSTEM.LIB

ftell (FS2)

```
long ftell(File * f);
```

DESCRIPTION

Return the current read/write position of the file. Bytes in a file are sequentially numbered starting at zero. If the current position is zero, then the first byte of the file will be read or written. If the position equals the file length, then no data can be read, but any write will append data to the file.

Note that no checking is done to see if the file descriptor s valid. If the File is not actually open, the return value will be random.

PARAMETERS

f	Pointer to file descriptor (initialized by **fopen_rd()**, **fopen_wr()** or **fcreate()**).

RETURN VALUE

Current read/write position (0 to length-of-file).

ERRNO VALUES

None.

LIBRARY

fs2.lib

SEE ALSO

fseek (FS2)

fshift

```
int fshift(File *f, int count, char *buffer);
```

DESCRIPTION

Removes **count** number of bytes from the beginning of a file and copies them to memory pointed to by **buffer**.

PARAMETERS

f	A pointer to the file.
count	Number of bytes to shift out.
***buffer**	Location to store shifted bytes. If this is **NULL**, the bytes will be discarded.

RETURN VALUE

Number of bytes shifted out: Success.
0: Error.

LIBRARY

FILESYSTEM.LIB

fwrite (FS1)

```
int fwrite(File* f, char* buf, int len);
```

DESCRIPTION

Appends **len** bytes from the source buffer to the end of the file.

PARAMETERS

f	A pointer to the file to write to.
buf	A pointer to the source buffer.
len	The number of bytes to write.

RETURN VALUE

The number of bytes written: Success.
0: Failure.

LIBRARY

FILESYSTEM.LIB

fwrite (FS2)

```
int fwrite(File* f, void* buf, int len);
```

DESCRIPTION

Write data to file opened for writing. The data is written starting at the "current position". This is zero (start of file) when it is opened or created, but may be changed by fread, fwrite, fshift or fseek functions. After writing the data, the current position is advanced to the position just after the last byte written. Thus, sequential calls to **fwrite()** will add or append data contiguously.

Unlike the previous file system (**FILESYSTEM.LIB**), this library allows files to be overwritten not just appended. Internally, overwrite and append are different operations with differing performance, depending on the underlying hardware. Generally, appending is more efficient especially with byte-writable flash memory. If the application allows, it is preferable to use append/shift rather than overwrite. In order to ensure that data is appended, use **fseek(f, 0, SEEK_END)** before calling **fwrite()**.

The same "current position" pointer is used for both read and write. If interspersing read and write, then **fseek()** should be used to ensure the correct position for each operation. Alternatively, the same file can be opened twice, with one descriptor used for read and the other for write. This precludes use of **fshift()**, since it does not tolerate shared files.

PARAMETERS

f	Pointer to file descriptor (initialized by **fopen_wr()** or **fcreate()**).
buf	Data buffer located in root data memory or stack.
len	Length of data (0 to 32767 inclusive).

RETURN VALUE

len: Success.
<len: Partial success. Returns amount successfully written. **errno** gives more details.
0: Failure, or **len** was zero.

ERRNO VALUES

EBADFD - File descriptor not opened, or is read-only.
EINVAL - **len** less than zero.
0 - Success, but **len** was zero.
EIO - I/O error.
ENOSPC - extent out of space.

LIBRARY

fs2.LIB

SEE ALSO

fread (FS2)

ftoa

```
int ftoa(float f, char *buf);
```

DESCRIPTION

Converts a float number to a character string.

The character string only displays the mantissa up to 9 digits, no decimal points, and a minus sign if **f** is negative. The function returns the exponent (of 10) that should be used to compensate for the string: **ftoa(1.0,buf)** yields **buf="100000000"**, and returns **-8**.

PARAMETERS

f	Float number to convert
buf	Converted string. The string is no longer than 10 characters long.

RETURN VALUE

The exponent of the number.

LIBRARY

STDIO.LIB

SEE ALSO

utoa, itoa

getchar

```
char getchar(void);
```

DESCRIPTION

Busy waits for a character to be typed from the stdio window in Dynamic C. The user should make sure only one process calls this function at a time.

RETURN VALUE

A character typed in the Stdio window in Dynamic C.

LIBRARY

STDIO.LIB

SEE ALSO

gets, putchar

getcrc

```
int getcrc(char *dataarray, char count, int accum);
```

DESCRIPTION

Computes the Cyclic Redundancy Check (CRC), or check sum, for **count** bytes (maximum 255) of data in buffer. Calls to **getcrc** can be "concatenated" using **accum** to compute the CRC for a large buffer.

PARAMETERS

dataarray	Data buffer
count	Number of bytes. Maximum is 255.
accum	Base CRC for the data array.

RETURN VALUE

CRC value.

LIBRARY

MATH.LIB

gets

```
char *gets(char *s);
```

DESCRIPTION

Waits for a string terminated by <CR> at the stdio window. The string returned is **NULL**-terminated without the return. The user should make sure only one process calls this function at a time.

PARAMETERS

s The input string is put to the location pointed to by the argument **s**. The caller is responsible to make sure the location pointed to by **s** is big enough for the string.

RETURN VALUE

Same pointer passed in, but string is changed to a **NULL**-terminated.

LIBRARY

STDIO.LIB

SEE ALSO

puts, getchar

GetVectExtern2000

```
unsigned GetVectExtern2000();
```

DESCRIPTION

Reads the address of external interrupt table entry. This function really just returns what is present in the table. The return value is meaningless if the address of the external interrupt has not been written.

RETURN VALUE

Jump address in vector table.

LIBRARY

SYS.LIB

SEE ALSO

GetVectIntern, SetVectExtern2000, SetVectIntern

GetVectExtern3000

```
unsigned GetVectExtern3000(int interruptNum);
```

DESCRIPTION

Function to read the address of an external interrupt table entry for the Rabbit 3000 CPU. This function really just returns whatever value is at the address:

(internal vector table base) + (vectNum*8) + 1.

PARAMETER

interruptNum Interrupt number. Should be 0 or 1.

RETURN VALUE

Jump address in vector table

SEE ALSO

SetVectExtern3000, SetVectIntern, GetVectIntern

GetVectIntern

```
unsigned GetVectIntern(int vectNum);
```

DESCRIPTION

Reads the address of the internal interrupt table entry and returns whatever value is at the address **(internal vector table base) + (vectNum*16) + 1**.

PARAMETER

vectNum Interrupt number; should be 0–15.

RETURN VALUE

Jump address in vector table.

LIBRARY

SYS.LIB

SEE ALSO

GetVectExtern2000, SetVectExtern2000, SetVectIntern

gps_get_position

```
int gps_get_position(GPSPositon *newpos, char *sentence);
```

DESCRIPTION

Parses a sentence to extract position data. This function is able to parse any of the following GPS sentence formats: GGA, GLL or RMC.

PARAMETERS

newpos	A **GPSPosition** structure to fill.
sentence	A string containing a line of GPS data. in NMEA-0183 format.

RETURN VALUE

0: Success.
-1: Parsing error.
-2: Sentence marked invalid.

LIBRARY

gps.lib

gps_get_utc

```
int gps_get_utc(struct tm *newtime, char *sentence);
```

DESCRIPTION

Parses an RMC sentence to extract time data.

PARAMETERS

newtime	**tm** structure to fill with new UTC time.
sentence	A string containing a line of GPS data in NMEA-0183 format (RMC sentence).

RETURN VALUE

0: Success.
-1: Parsing error.
-2: Sentence marked invalid.

LIBRARY

gps.lib

gps_ground_distance

```
float gps_ground_distance(GPSPosition *a, GPSPosition *b);
```

DESCRIPTION

Calculates ground distance (in km) between to geographical points. (Uses spherical earth model.)

PARAMETERS

a	First point.
b	Second point.

RETURN VALUE

Distance in kilometers.

LIBRARY

gps.lib

hanncplx

```
void hanncplx(int *x, int N, int *blockexp);
```

DESCRIPTION

Convolves an **N**-point complex spectrum with the three-point Hann kernel. The filtered spectrum replaces the original spectrum.

The function produces the same results as would be obtained by multiplying the corresponding time sequence by the Hann raised-cosine window.

The zero–crossing width of the main lobe produced by the Hann window is 4 DFT bins. The adjacent sidelobes are 32 db below the main lobe. Sidelobes decay at an asymptotic rate of 18 db per octave.

N must be a power of 2 and lie between 4 and 1024. An invalid **N** causes a RANGE exception.

PARAMETERS

x	Pointer to **N**-element array of complex fractions.
N	Number of complex elements in array **x**.
blockexp	Pointer to integer block exponent.

LIBRARY

FFT.LIB

SEE ALSO

fftcplx, fftcplxinv, fftreal, fftrealinv, hanncplx, powerspectrum

hannreal

```
void hannreal(int *x, int N, int *blockexp);
```

DESCRIPTION

Convolves an **N**-point positive-frequency complex spectrum with the three-point Hann kernel. The function produces the same results as would be obtained by multiplying the corresponding time sequence by the Hann raised-cosine window.

The zero–crossing width of the main lobe produced by the Hann window is 4 DFT bins. The adjacent sidelobes are 32 db below the main lobe. Sidelobes decay at an asymptotic rate of 18 db per octave.

The imaginary part of the dc term (stored in **x[1]**) is considered to be the real part of the *fmax* term. The dc and *fmax* spectral components take part in the convolution along with the other spectral components. The real part of *fmax* component affects the real part of the *X*[*N*-1] component (and vice versa), and should not arbitrarily be set to zero unless these components are unimportant.

PARAMETERS

x	Pointer to **N**-element array of complex fractions.
N	Number of complex elements in array **x**.
blockexp	Pointer to integer block exponent.

RETURN VALUE

None. The filtered spectrum replaces the original spectrum.

LIBRARY

FFT.LIB

SEE ALSO

fftcplx, fftcplxinv, fftreal, fftrealinv, hanncplx, powerspectrum

HDLCdropX

```
int HDLCdropX(); /* Where X is E or F */
```

DESCRIPTION

Drops the next received packet, freeing up its buffer. This must be used if the packet has been examined with **HDLCpeekX()** and is no longer needed. A call to **HDLCreveiceX()** is the only other way to free up the buffer.

This function is intended for use with the Rabbit 3000 microprocessor.

RETURN VALUE

1: Packet dropped.
0: No received packets were available.

LIBRARY

HDLC_PACKET.LIB

HDLCerrorX

```
int HDLCerrorX(unsigned long *bufptr, int *lenptr);
   /* Where X is E or F */
```

DESCRIPTION

This function returns a set of possible error flags as an integer. A received packet with errors is automatically dropped.

Masks are used to check which errors have occurred. The masks are:

- **HDLC_NOBUFFER** - driver ran out of buffers for received packets.
- **HDLC_OVERRUN** - a byte was overwritten and lost before the ISR could retreive it.
- **HDLC_OVERFLOW** - a received packet was too long for the buffers.
- **HDLC_ABORTED** - a received packet was aborted by the sender during tranmission.
- **HDLC_BADCRC** - a packet with an incorrect CRC was received.

This function is intended for use with the Rabbit 3000 microprocessor.

RETURN VALUE

Error flags (see above)

LIBRARY

HDLC_PACKET.LIB

HDLCopenX

```
int HDLCopenX(long baud, char encoding, unsigned long buffers,
   int buffer_count, int buffer_size); /* Where X is E or F */
```

DESCRIPTION

Opens serial port E or F in HDLC mode. Sets up buffers to hold received packets. This function is intended for use with the Rabbit 3000 microprocessor.

PARAMETERS

baud The baud rate for the serial port. Due to imitations in the baud generator, non-standard baud rates will be approximated within 5% of the value requested.

encoding The bit encoding mode to use. Macro labels for the available options are:

- **HDLC_NRZ**
- **HDLC_NRZI**
- **HDLC_MANCHESTER**
- **HDLC_BIPHASE_SPACE**
- **HDLC_BIPHASE_MARK**

See the HDLC documentation for more detail on these modes.

buffers A pointer to the start of the extended momory block containing the receive buffers. This block must be allocated beforehand by the user. The size of the block should be:

```
(# of buffers) * ((size of buffer) + 4)
```

buffer_count The number of buffers in the block pointed to by **buffer**.

buffer_size The capacity of each buffer in the block pointed to by **buffer**.

RETURN VALUE

1 if the actual baud rate is within 5% of the requested baud rate,
0 otherwise.

LIBRARY

HDLC_PACKET.LIB

HDLCpeekX

```
int HDLCpeekX(unsigned long *bufptr, int *lenptr);
   /* Where X is E or F */
```

DESCRIPTION

Reports the location and size of the next available received packet if one is available. This function can be used to effeciently inspect a received packet without actually copying it into a root memory buffer. Once inspected, the buffer can be received normally (see **HDLCreceiveX()**), or dropped (see **HDLCdropX()**).

This function is intended for use with the Rabbit 3000 microprocessor.

PARAMETERS

bufptr	Pointer to location in xmem of the received packet.
lenptr	Pointer to the size of the received packet.

RETURN VALUE

1: The pointers **bufptr** and **lenptr** have been set for the received packet.
0: No received packets available.

LIBRARY

HDLC_PACKET.LIB

HDLCreceiveX

```
int HDLCreceiveX (char *rx_buffer, int length);
   /* Where X is E or F */
```

DESCRIPTION

Copies a received packet into **rx_buffer** if there is one. Packets are received in the order they arrive, even if multiple packets are currently stored in buffers.

This function is intended for use with the Rabbit 3000 microprocessor.

PARAMETERS

rx_buffer	Pointer to the buffer to copy a received packet into.
length	Size of the buffer pointed to by **rx_buffer**.

RETURN VALUE

≥**0**: Size of received packet.
-1: No packets are available to receive.
-2: The buffer is not large enough for the received packet. In this case, the packet remains in the receive buffer)

LIBRARY

HDLC_PACKET.LIB

HDLCsendX

```
int HDLCsendX(char *tx_buffer, int length); /* Where X is E or F */
```

DESCRIPTION

Transmits a packet out serial port E or F in HDLC mode. The tx_buffer is read directly while transmitting, therefore it cannot be altered until a subsequent call to **HDLCsendingX()** returns false, indicating that the driver is done with it.

This function is intended for use with the Rabbit 3000 microprocessor.

PARAMETERS

tx_buffer	A pointer to the packet to be sent. This buffer must not change while tranmitting (see above.)
length	The size of the buffer (in bytes).

RETURN VALUE

1: Sending packet
0: Cannot send, another packet is currently being transmitted.

LIBRARY

HDLC_PACKET.LIB

HDLCsendingX

```
int HDLCsendingX(); /* Where X is E or F */
```

DESCRIPTION

Returns true if a packet is currently being transmitted. This function is intended for use with the Rabbit 3000 microprocessor.

RETURN VALUE

1: Currently sending a packet.
0: Transmitter is idle.

LIBRARY

HDLC_PACKET.LIB

hitwd

```
void hitwd();
```

DESCRIPTION

Hits the watchdog timer, postponing a hardware reset for 2 seconds. Unless the watchdog timer is disabled, a program must call this function periodically, or the controller will automatically reset itself. If the virtual driver is enabled (which it is by default), it will call **hitwd** in the background. The virtual driver also makes additional "virtual" watchdog timers available.

LIBRARY

VDRIVER.LIB

htoa

```
char *htoa(int value, char *buf);
```

DESCRIPTION

Converts integer **value** to hexadecimal number and puts result into **buf**.

PARAMETERS

value	16-bit number to convert
buf	Character string of converted number

RETURN VALUE

Pointer to end (**NULL** terminator) of string in **buf**.

LIBRARY

STDIO.LIB

SEE ALSO

itoa, utoa, ltoa

IntervalMs

```
int IntervalMs( long ms );
```

DESCRIPTION

Similar to **DelayMs** but provides a periodic delay based on the time from the previous call. Intended for use with **waitfor**.

PARAMETERS

ms	The number of milliseconds to wait.

RETURN VALUE

0: Not finished.
1: Delay has expired.

LIBRARY

COSTATE.LIB

IntervalSec

```
int IntervalSec( long sec );
```

DESCRIPTION

Similar to **DelayMs** but provides a periodic delay based on the time from the previous call. Intended for use with **waitfor**.

PARAMETERS

sec	The number of seconds to delay.

RETURN VALUE

0: Not finished.
1: Delay has expired.

LIBRARY

COSTATE.LIB

IntervalTick

```
int IntervalTick( long tick );
```

DESCRIPTION

Provides a periodic delay based on the time from the previous call. Intended for use with **waitfor**. A tick is 1/1024 seconds.

PARAMETERS

tick The number of ticks to delay

RETURN VALUE

0: Not finished.
1: Delay has expired.

LIBRARY

COSTATE.LIB

ipres

```
void ipres(void);
```

DESCRIPTION

Dynamic C expands this call inline. Restore previous interrupt priority by rotating the IP register.

LIBRARY

UTIL.LIB

SEE ALSO

ipset

ipset

```
void ipset(int priority);
```

DESCRIPTION

Dynamic C expands this call inline. Replaces current interrupt priority with another by rotating the new priority into the IP register.

PARAMETERS

priority Interrupt priority range 0–3, lowest to highest priority.

LIBRARY

UTIL.LIB

SEE ALSO

ipres

isalnum

```
int isalnum(int c);
```

DESCRIPTION

Tests for an alphabetic or numeric character, (A to Z, a to z and 0 to 9).

PARAMETERS

c Character to test.

RETURN VALUE

0 if not an alphabetic or numeric character.
!0 otherwise.

LIBRARY

STRING.LIB

SEE ALSO

isalpha, isdigit, ispunct

isalpha

```
int isalpha(int c);
```

DESCRIPTION

Tests for an alphabetic character, (A to Z, or a to z).

PARAMETERS

c Character to test.

RETURN VALUE

0 if not a alphabetic character.
!0 otherwise.

LIBRARY

STRING.LIB

SEE ALSO

isalnum, isdigit, ispunct

iscntrl

```
int iscntrl(int c);
```

DESCRIPTION

Tests for a control character: 0 <= **c** <= 31 or **c** == 127.

PARAMETERS

c Character to test.

RETURN VALUE

0 if not a control character.
!0 otherwise.

LIBRARY

STRING.LIB

SEE ALSO

isalpha, isalnum, isdigit, ispunct

isCoDone

```
int isCoDone(CoData *p);
```

DESCRIPTION

Determine if costatement is initialized and not running.

PARAMETERS

p Address of costatement

RETURN VALUE

1 if costatement is initialized and not running.
0 otherwise.

LIBRARY

COSTATE.LIB

isCoRunning

```
int isCoRunning(CoData *p);
```

DESCRIPTION

Determine if costatement is stopped or running.

PARAMETERS

p Address of costatement

RETURN VALUE

1 if costatement is running.
0 otherwise.

LIBRARY

COSTATE.LIB

isdigit

```
int isdigit(int c);
```

DESCRIPTION

Tests for a decimal digit: 0 - 9

PARAMETERS

c Character to test.

RETURN VALUE

0 if not a decimal digit.
!0 otherwise.

LIBRARY

STRING.LIB

SEE ALSO

isxdigit, isalpha, isalpha

isgraph

```
int isgraph(int c);
```

DESCRIPTION

Tests for a printing character other than a space: 33 <= **c** <= 126

PARAMETERS

c Character to test.

RETURN VALUE

0: **c** is not a printing character.
!0: **c** is a printing character.

LIBRARY

STRING.LIB

SEE ALSO

isprint, isalpha, isalnum, isdigit, ispunct

islower

```
int islower(int c);
```

DESCRIPTION

Tests for lower case character.

PARAMETERS

c Character to test.

RETURN VALUE

0 if not a lower case character.
!0 otherwise.

LIBRARY

STRING.LIB

SEE ALSO

tolower, toupper, isupper

isspace

```
int isspace(int c);
```

DESCRIPTION

Tests for a white space, character, tab, return, newline, vertical tab, form feed, and space: 9 <= **c** <= 13 and **c** == 32 .

PARAMETERS

c Character to test.

RETURN VALUE

0 if not, **!0** otherwise.

LIBRARY

STRING.LIB

SEE ALSO

ispunct

isprint

```
int isprint(int c);
```

DESCRIPTION

Tests for printing character, including space: 32 <= **c** <= 126

PARAMETERS

c Character to test.

RETURN VALUE

0 if not a printing character, **!0** otherwise.

LIBRARY

STRING.LIB

SEE ALSO

isdigit, isxdigit, isalpha, ispunct, isspace, isalnum, isgraph

ispunct

```
int ispunct(int c);
```

DESCRIPTION

Tests for a punctuation character.

Character	Decimal Code
space	32
! " # $ % & ' ( ) * + , - . /	33 <= **c** <= 47
: ; < = > ? @	58 <= **c** <= 64
[\] ^ _ `	91 <= **c** <= 96
{|} ~	123 <= **c** <= 126

PARAMETERS

c Character to test.

RETURN VALUE

0 if not a character.
!0 otherwise.

LIBRARY

STRING.LIB

SEE ALSO

isspace

isupper

```
int isupper(int c);
```

DESCRIPTION

Tests for upper case character.

PARAMETERS

c Character to test.

RETURN VALUE

0 if not, **!0** otherwise.

LIBRARY

STRING.LIB

SEE ALSO

tolower, toupper, islower

isxdigit

```
int isxdigit(int c);
```

DESCRIPTION

Tests for a hexadecimal digit: 0 - 9, A - F, a - f

PARAMETERS

c Character to test.

RETURN VALUE

0 if not a hexadecimal digit, **!0** otherwise.

LIBRARY

STRING.LIB

SEE ALSO

isdigit, isalpha, isalpha

itoa

```
char *itoa(int value, char *buf);
```

DESCRIPTION

Places up to a 5-digit character string, with a minus sign in the leftmost digit when appropriate, at ***buf**. The string represents **value**, a signed number.

Leading zeros are suppressed in the character string, except for one zero digit when **value** = 0. The longest possible string is "-32768."

PARAMETERS

value	16-bit signed number to convert
buf	Character string of converted number in base 10

RETURN VALUE

Pointer to the end (**NULL** terminator) of the string in **buf**.

LIBRARY

STDIO.LIB

SEE ALSO

atoi, utoa, ltoa

i2c_check_ack

```
int i2c_check_ack();
```

DESCRIPTION

Checks if slave pulls data low for ACK on clock pulse. Allows for clocks stretching on SCL going high.

RETURN VALUE

0: ACK sent from slave.
1: NAK sent from slave.
-1: Timeout occurred.

LIBRARY

I2C.LIB

SEE ALSO

Technical Note 215, *Using the I2C Bus with a Rabbit Microprocessor.*

i2c_init

```
void i2c_init();
```

DESCRIPTION

Sets up the SCL and SDA port pins for open-drain output.

LIBRARY

I2C.LIB

SEE ALSO

Technical Note 215, *Using the I2C Bus with a Rabbit Microprocessor.*

i2c_read_char

```
int i2c_read_char(char *ch);
```

DESCRIPTION

Reads 8 bits from the slave. Allows for clocks stretching on all SCL going high. This is not in the protocol for I^2C, but allows I^2C slaves to be implemented on slower devices.

PARAMETERS

ch	A one character return buffer.

RETURN VALUE

0: Success.
-1: Clock stretching timeout.

LIBRARY

I2C.LIB

SEE ALSO

Technical Note 215, *Using the I2C Bus with a Rabbit Microprocessor.*

i2c_send_ack

```
int i2c_send_ack();
```

DESCRIPTION

Sends ACK sequence to slave. ACK is usually sent after a successful transfer, where more bytes are going to be read.

RETURN VALUE

0: Success.
-1: Clock stretching timeout.

LIBRARY

I2C.LIB

SEE ALSO

Technical Note 215, *Using the I2C Bus with a Rabbit Microprocessor.*

i2c_send_nak

```
int i2c_send_nak();
```

DESCRIPTION

Sends NAK sequence to slave. NAK is often sent when the transfer is finished.

RETURN VALUE

0: Success.
-1: Clock stretching timeout.

LIBRARY

I2C.LIB

SEE ALSO

Technical Note 215, *Using the I2C Bus with a Rabbit Microprocessor.*

i2c_start_tx

```
int i2c_start_tx();
```

DESCRIPTION

Initiates I^2C transmission by sending the start sequence, which is defined as a high to low transition on SDA while SCL is high. The point being that SDA is supposed to remain stable while SCL is high. If it does not, then that indicates a start (S) or stop (P) condition. This function first waits for possible clock stretching, which is when a bus peripheral holds SCK low.

RETURN VALUE

0: Success.
-1: Clock stretching timeout.

LIBRARY

I2C.LIB

SEE ALSO

Technical Note 215, *Using the I2C Bus with a Rabbit Microprocessor.*

i2c_startw_tx

```
int i2c_startw_tx();
```

DESCRIPTION

Initiates I^2C transmission by sending the start sequence, which is defined as a high to low transition on SDA while SCL is high. The point being that SDA is supposed to remain stable while SCL is high. If it does not, then that indicates a start (S) or stop (P) condition. This function first waits for possible clock stretching, which is when a bus peripheral holds SCK low.

This function is essentially the same as **i2c_start_tx()** with the addition of a clock stretch delay, which is 2000 "counts," inserted after the start sequence. (A count is an iteration through a loop.)

RETURN VALUE

0: Success.
-1: Clock stretching timeout.

LIBRARY

I2C.LIB

SEE ALSO

Technical Note 215, *Using the I2C Bus with a Rabbit Microprocessor.*

i2c_stop_tx

```
void i2c_stop_tx();
```

DESCRIPTION

Sends the stop sequence to the slave, which is defined as bringing SDA high while SCL is high, i.e., the clock goes high, then data goes high.

LIBRARY

I2C.LIB

SEE ALSO

Technical Note 215, *Using the I2C Bus with a Rabbit Microprocessor.*

i2c_write_char

```
int i2c_write_char(char d);
```

DESCRIPTION

Sends 8 bits to slave. Checks if slave pulls data low for ACK on clock pulse. Allows for clocks stretching on SCL going high.

PARAMETERS

d	Character to send

RETURN VALUE

0: Success.
-1: Clock stretching timeout.
1: NAK sent from slave.

LIBRARY

I2C.LIB

SEE ALSO

Technical Note 215, *Using the I2C Bus with a Rabbit Microprocessor.*

kbhit

```
int kbhit();
```

DESCRIPTION

Detects keystrokes in the Dynamic C Stdio window.

RETURN VALUE

!0 if a key has been pressed, **0** otherwise.

LIBRARY

UTIL.LIB

labs

```
long labs(long x);
```

DESCRIPTION

Computes the long integer absolute value of long integer **x**.

PARAMETERS

x	Number to compute.

RETURN VALUE

x, if **x** ≥ 0, else **-x**.

LIBRARY

MATH.LIB

SEE ALSO

abs, fabs

ldexp

```
float ldexp(float x, int n);
```

DESCRIPTION

Computes **x*(2**n)**

PARAMETERS

x	The value between 0.5 inclusive, and 1.0
n	An integer

RETURN VALUE

The result of **x*(2^n)**.

LIBRARY

MATH.LIB

SEE ALSO

frexp, exp

log

```
float log(float x);
```

DESCRIPTION

Computes the logarithm, base e, of real **float** value **x**.

PARAMETERS

x	Float value

RETURN VALUE

The function returns –INF and signals a domain error when **x** ≤ 0.

LIBRARY

MATH.LIB

SEE ALSO

exp, log10

log10

```
float log10(float x);
```

DESCRIPTION

Computes the base 10 logarithm of real **float** value **x**.

PARAMETERS

x	Value to compute

RETURN VALUE

The log base 10 of **x**.

The function returns –INF and signals a domain error when **x** ≤ 0.

LIBRARY

MATH.LIB

SEE ALSO

log, exp

longjmp

```
void longjmp(jmp_buf env, int val);
```

DESCRIPTION

Restores the stack environment saved in array **env[ ]**. See the description of **setjmp** for details of use.

PARAMETERS

env	Environment previously saved with **setjmp**.
val	Integer result of **setjmp**.

LIBRARY

SYS.LIB

SEE ALSO

setjmp

loophead

```
void loophead();
```

DESCRIPTION

This function should be called within the main loop in a program. It is necessary for proper single-user cofunction abandonment handling.

When two costatements are requesting access to a single-user cofunction, the first request is honored and the second request is held. When **loophead()** notices that the first caller is not being called each time around the loop, it cancels the request, calls the abandonment code and allows the second caller in.

See **Samples\Cofunc\Cofaband.c** for sample code showing abandonment handling.

PARAMETERS

None

LIBRARY

COFUNC.LIB

loopinit

```
void loopinit();
```

DESCRIPTION

This function should be called in the beginning of a program that uses single-user co-functions. It initializes internal data structures that are used by **loophead()**.

PARAMETERS

None

LIBRARY

COFUNC.LIB

ltoa

```
char *ltoa(long num, char *ibuf)
```

DESCRIPTION

This function outputs a signed long number to the character array.

PARAMETERS

num	Signed long number
ibuf	Pointer to character array

RETURN VALUE

Pointer to the same array passed in to hold the result.

LIBRARY

STDIO.LIB

SEE ALSO

ltoa

ltoan

```
int ltoan(long num);
```

DESCRIPTION

This function returns the number of characters required to display a signed long number.

PARAMETERS

num 32-bit signed number

RETURN VALUE

The number of characters to display signed long number.

LIBRARY

STDIO.LIB

SEE ALSO

ltoa

lx_format

```
int lx_format(FSLXnum lxn, long wearlevel);
```

DESCRIPTION

Format a specified file system extent. This must not be called before calling **fs_init()**. All files which have either or both metadata and data on this extent are deleted. Formatting can be quite slow (depending on hardware) so it is best performed after power-up, if at all.

PARAMETERS

lxn	Logical extent number (1.._fs.num_lx inclusive).
wearlevel	Initial wearlevel value. This should be 1 if you have a new flash, and some larger number if the flash is used. If you are reformatting a flash, you can use 0 to use the old flash wear levels.

RETURN VALUE

0: Success.
!0: Failure.

ERRNO VALUES

ENODEV - no such extent number, or extent is reserved.
EBUSY - one or more files were open on this extent.
EIO - I/O error during format. If this occurs, retry the format operation. If it fails again, there is probably a hardware error.

LIBRARY

fs2.LIB

SEE ALSO

fs_init, fs_format

md5_append

```
void md5_append(md5_state_t *pms, char *data, int nbytes);
```

DESCRIPTION

This function will take a buffer and compute the MD5 hash of its contents, combined with all previous data passed to it. This function can be called several times to generate the hash of a large amount of data

PARAMETERS

md5_append	Pointer to the **md5_state_t** structure that was initialized by **md5_init**.
data	Pointer to the data to be hashed.
nbytes	Length of the data to be hashed.

LIBRARY

MD5.LIB

md5_init

```
void md5_init(md5_state_t *pms);
```

DESCRIPTION

Initialize the MD5 hash process. Initial values are generated for the structure, and this structure will identify a particular transaction in all subsequent calls to the md5 library.

PARAMETER

pms	Pointer to the **md5_state_t** structure.

LIBRARY

MD5.LIB

md5_finish

```
void md5_finish(md5_state_t *pms, char digest[16]);
```

DESCRIPTION

Completes the hash of all the received data and generates the final hash value.

PARAMETERS

pms	Pointer to the **md5_state_t** structure that was initialized by **md5_init**.
digest	The 16-byte array that the hash value will be written into.

LIBRARY

MD5.LIB

memchr

```
void *memchr(void *src, int ch, unsigned int n);
```

DESCRIPTION

Searches up to **n** characters at memory pointed to by **src** for character **ch**.

PARAMETERS

src	Pointer to memory source.
ch	Character to search for.
n	Number of bytes to search.

RETURN VALUE

Pointer to first occurrence of **ch** if found within **n** characters. Otherwise returns **NULL**.

LIBRARY

STRING.LIB

SEE ALSO

strrchr, strstr

memcmp

```
int memcmp(void *s1, void *s2, size_t n);
```

DESCRIPTION

Performs unsigned character by character comparison of two memory blocks of length **n**.

PARAMETERS

s1	Pointer to block 1.
s2	Pointer to block 2.
n	Maximum number of bytes to compare.

RETURN VALUE

<0: A character in **str1** is less than the corresponding character in **str2**.
0: **str1** is identical to **str2**.
>0: A character in **str1** is greater than the corresponding character in **str2**.

LIBRARY

STRING.LIB

SEE ALSO

strncmp

memcpy

```
void *memcpy(void *dst, void *src, unsigned int n);
```

DESCRIPTION

Copies a block of bytes from one destination to another. Overlap is handled correctly.

PARAMETERS

dst	Pointer to memory destination
src	Pointer to memory source
n	Number of characters to copy

RETURN VALUE

Pointer to destination.

LIBRARY

STRING.LIB

SEE ALSO

memmove, memset

memmove

```
void *memmove(void *dst, void *src, unsigned int n);
```

DESCRIPTION

Copies a block of bytes from one destination to another. Overlap is handled correctly.

PARAMETERS

dst	Pointer to memory destination
src	Pointer to memory source
n	Number of characters to copy

RETURN VALUE

Pointer to destination.

LIBRARY

STRING.LIB

SEE ALSO

memcpy, memset

memset

```
void *memset(void *dst, int chr, unsigned int n);
```

DESCRIPTION

Sets the first **n** bytes of a block of memory to byte destination.

PARAMETERS

dst	Block of memory to set
chr	Byte destination
n	Amount of bytes to set

LIBRARY

STRING.LIB

mktime

```
unsigned long mktime(struct tm *timeptr);
```

DESCRIPTION

Converts the contents of structure pointed to by **timeptr** into seconds.

```
struct tm {
   char tm_sec;       // seconds 0-59
   char tm_min;       // 0-59
   char tm_hour;      // 0-23
   char tm_mday;      // 1-31
   char tm_mon;       // 1-12
   char tm_year;      // 80-147 (1980-2047)
   char tm_wday;      // 0-6 0==sunday
};
```

PARAMETERS

timeptr	Pointer to **tm** structure

RETURN VALUE

Time in seconds since January 1, 1980.

LIBRARY

RTCLOCK.LIB

SEE ALSO

mktm, tm_rd, tm_wr

mktm

```
unsigned int mktm(struct tm *timeptr, unsigned long time);
```

DESCRIPTION

Converts the seconds (**time**) to date and time and fills in the fields of the **tm** structure with the result.

```
struct tm {
   char tm_sec;       // seconds 0-59
   char tm_min;       // 0-59
   char tm_hour;      // 0-23
   char tm_mday;      // 1-31
   char tm_mon;       // 1-12
   char tm_year;      // 80-147 (1980-2047)
   char tm_wday;      // 0-6 0==sunday
};
```

PARAMETERS

timeptr Address to store date and time into structure:

time Seconds since January 1, 1980.

RETURN VALUE

0

LIBRARY

RTCLOCK.LIB

SEE ALSO

mktime, tm_rd, tm_wr

modf

```
float modf(float x, int *n);
```

DESCRIPTION

Splits **x** into a fraction and integer, **f + n**.

PARAMETERS

x	Floating-point integer
n	An integer

RETURN VALUE

The integer part in ***n** and the fractional part satisfies **|f| < 1.0**

LIBRARY

MATH.LIB

SEE ALSO

fmod, ldexp

OS_ENTER_CRITICAL

```
void OS_ENTER_CRITICAL();
```

DESCRIPTION

Enter a critical section. Interrupts will be disabled until **OS_EXIT_CRITICAL()** is called. Task switching is disabled. This function must be used with great care, since misuse can greatly increase the latency of your application. Note that nesting **OS_ENTER_CRITICAL()** calls will work correctly.

LIBRARY

UCOS2.LIB

OS_EXIT_CRITICAL

```
void OS_EXIT_CRITICAL();
```

DESCRIPTION

Exit a critical section. If the corresponding previous **OS_ENTER_CRITICAL()** call disabled interrupts (that is, interrupts were not already disabled), then interrupts will be enabled. Otherwise, interrupts will remain disabled. Hence, nesting calls to **OS_ENTER_CRITICAL()** will work correctly.

LIBRARY

UCOS2.LIB

OSInit

```
void OSInit(void);
```

DESCRIPTION

Initializes µC/OS-II data; must be called before any other µC/OS-II functions are called.

LIBRARY

UCOS2.LIB

SEE ALSO

OSTaskCreate, OSTaskCreateExt, OSStart

OSMboxAccept

```
void *OSMboxAccept (OS_EVENT *OSMboxAccept);
```

DESCRIPTION

Checks the mailbox to see if a message is available. Unlike **OSMboxPend()**, **OSMboxAccept()** does not suspend the calling task if a message is not available.

PARAMETERS

OSMboxAccept Pointer to the mailbox's event control block.

RETURN VALUE

Pointer to available message, or a **NULL** pointer if there is no available message or an error condition exists.

LIBRARY

UCOS2.LIB

SEE ALSO

OSMboxCreate, OSMboxPend, OSMboxPost, OSMboxQuery

OSMboxCreate

```
OS_EVENT *OSMboxCreate (void *msg);
```

DESCRIPTION

Creates a message mailbox if event control blocks are available.

PARAMETERS

msg	Pointer to a message to put in the mailbox.

RETURN VALUE

Pointer to mailbox's event control block, or **NULL** pointer if no event control block was available.

LIBRARY

UCOS2.LIB

SEE ALSO

OSMboxAccept, OSMboxPend, OSMboxPost, OSMboxQuery

OSMboxPend

```
void *OSMboxPend(OS_EVENT *pevent, INT16U timeout, INT8U *err);
```

DESCRIPTION

Waits for a message to be sent to a mailbox.

PARAMETERS

pevent	Pointer to mailbox's event control block.
timeout	Allows task to resume execution if a message was not received by the number of clock ticks specified. Specifying 0 means the task is willing to wait forever.
err	Pointer to a variable for holding an error code.

RETURN VALUE

Pointer to a message or, if a timeout or error condition occurs, a **NULL** pointer.

LIBRARY

UCOS2.LIB

SEE ALSO

OSMboxAccept, OSMboxCreate, OSMboxPost, OSMboxQuery

OSMboxPost

```
INT8U OSMboxPost (OS_EVENT *pevent, void *msg);
```

DESCRIPTION

Sends a message to the specified mailbox

PARAMETERS

pevent	Pointer to mailbox's event control block.
msg	Pointer to message to be posted. A **NULL** pointer must not be sent.

RETURN VALUE

OS_NO_ERR	The call was successful and the message was sent.
OS_MBOX_FULL	The mailbox already contains a message. Only one message at a time can be sent and thus, the message MUST be consumed before another can be sent.
OS_ERR_EVENT_TYPE	Attempting to post to a non-mailbox.

LIBRARY

UCOS2.LIB

SEE ALSO

OSMboxAccept, OSMboxCreate, OSMboxPend, OSMboxQuery

OSMboxQuery

```
INT8U OSMboxQuery (OS_EVENT *pevent, OS_MBOX_DATA *pdata);
```

DESCRIPTION

Obtains information about a message mailbox.

PARAMETERS

pevent	Pointer to message mailbox's event control block.
pdata	Pointer to a data structure for information about the message mailbox

RETURN VALUE

OS_NO_ERR	The call was successful and the message was sent.
OS_ERR_EVENT_TYPE	Attempting to obtain data from a non mailbox.

LIBRARY

UCOS2.LIB

SEE ALSO

OSMboxAccept, OSMboxCreate, OSMboxPend, OSMboxPost

OSMemCreate

```
OS_MEM *OSMemCreate (void *addr, INT32U nblks, INT32U blksize,
   INT8U *err);
```

DESCRIPTION

Creates a fixed-sized memory partition that will be managed by µC/OS-II.

PARAMETERS

addr	Pointer to starting address of the partition.
nblks	Number of memory blocks to create in the partition.
blksize	The size (in bytes) of the memory blocks.
err	Pointer to variable containing an error message.

RETURN VALUE

Pointer to the created memory partition control block if one is available, **NULL** pointer otherwise.

LIBRARY

UCOS2.LIB

SEE ALSO

OSMemGet, OSMemPut, OSMemQuery

OSMemGet

```
void *OSMemGet (OS_MEM *pmem, INT8U *err);
```

DESCRIPTION

Gets a memory block from the specified partition.

PARAMETERS

pmem	Pointer to partition's memory control block
err	Pointer to variable containing an error message

RETURN VALUE

Pointer to a memory block or a **NULL** pointer if an error condition is detected.

LIBRARY

UCOS2.LIB

SEE ALSO

OSMemCreate, OSMemPut, OSMemQuery

OSMemPut

```
INT8U OSMemPut(OS_MEM *pmem, void *pblk);
```

DESCRIPTION

Returns a memory block to a partition.

PARAMETERS

pmem	Pointer to the partition's memory control block.
pblk	Pointer to the memory block being released.

RETURN VALUE

OS_NO_ERR	The memory block was inserted into the partition.
OS_MEM_FULL	If returning a memory block to an already FULL memory partition. (More blocks were freed than allocated!)

LIBRARY

UCOS2.LIB

SEE ALSO

OSMemCreate, OSMemGet, OSMemQuery

OSMemQuery

```
INT8U OSMemQuery (OS_MEM *pmem, OS_MEM_DATA *pdata);
```

DESCRIPTION

Determines the number of both free and used memory blocks in a memory partition.

PARAMETERS

pmem	Pointer to partition's memory control block.
pdata	Pointer to structure for holding information about the partition.

RETURN VALUE

OS_NO_ERR	This function always returns no error.

LIBRARY

UCOS2.LIB

SEE ALSO

OSMemCreate, OSMemGet, OSMemPut

OSQAccept

```
void *OSQAccept (OS_EVENT *pevent);
```

DESCRIPTION

Checks the queue to see if a message is available. Unlike **OSQPend()**, with **OSQAccept()** the calling task is not suspended if a message is unavailable.

PARAMETERS

pevent	Pointer to the message queue's event control block.

RETURN VALUE

Pointer to message in the queue if one is available, **NULL** pointer otherwise.

LIBRARY

UCOS2.LIB

SEE ALSO

OSQCreate, OSQFlush, OSQPend, OSQPost, OSQPostFront, OSQQuery

OSQCreate

```
OS_EVENT *OSQCreate (void **start, INT16U qsize);
```

DESCRIPTION

Creates a message queue if event control blocks are available.

PARAMETERS

start	Pointer to the base address of the message queue storage area. The storage area MUST be declared an array of pointers to void: **void *MessageStorage[qsize].**
qsize	Number of elements in the storage area.

RETURN VALUE

Pointer to message queue's event control block or **NULL** pointer if no event control blocks were available.

LIBRARY

UCOS2.LIB

SEE ALSO

OSQAccept, OSQFlush, OSQPend, OSQPost, OSQPostFront, OSQQuery

OSQFlush

```
INT8U OSQFlush (OS_EVENT *pevent);
```

DESCRIPTION

Flushes the contents of the message queue.

PARAMETERS

pevent	Pointer to message queue's event control block.

RETURN VALUE

OS_NO_ERR: Success.
OS_ERR_EVENT_TYPE: A pointer to a queue was not passed.
OS_ERR_PEVENT_NULL: If **pevent** is a **NULL** pointer.

LIBRARY

UCOS2.LIB

SEE ALSO

OSQAccept, OSQCreate, OSQPend, OSQPost, OSQPostFront, OSQQuery

OSQPend

```
void *OSQPend (OS_EVENT *pevent, INT16U timeout, INT8U *err);
```

DESCRIPTION

Waits for a message to be sent to a queue.

PARAMETERS

pevent	Pointer to message queue's event control block.
timeout	Allow task to resume execution if a message was not received by the number of clock ticks specified. Specifying 0 means the task is willing to wait forever.
err	Pointer to a variable for holding an error code.

RETURN VALUE

Pointer to a message or, if a timeout occurs, a **NULL** pointer.

LIBRARY

UCOS2.LIB

SEE ALSO

OSQAccept, OSQCreate, OSQFlush, OSQPost, OSQPostFront, OSQQuery

OSQPost

```
INT8U OSQPost (OS_EVENT *pevent, void *msg);
```

DESCRIPTION

Sends a message to the specified queue.

PARAMETERS

pevent	Pointer to message queue's event control block.
msg	Pointer to the message to send. **NULL** pointer must not be sent.

RETURN VALUE

OS_NO_ERR: The call was successful and the message was sent.
OS_Q_FULL: The queue cannot accept any more messages because it is full.
OS_ERR_EVENT_TYPE: If a pointer to a queue not passed.
OS_ERR_PEVENT_NULL: If **pevent** is a **NULL** pointer.
OS_ERR_POST_NULL_PTR: If attempting to post to a **NULL** pointer.

LIBRARY

UCOS2.LIB

SEE ALSO

OSQAccept, OSQCreate, OSQFlush, OSQPend, OSQPostFront, OSQQuery

OSQPostFront

```
INT8U OSQPostFront (OS_EVENT *pevent, void *msg);
```

DESCRIPTION

Sends a message to the specified queue, but unlike **OSQPost()**, the message is posted at the front instead of the end of the queue. Using **OSQPostFront()** allows 'priority' messages to be sent.

PARAMETERS

pevent	Pointer to message queue's event control block.
msg	Pointer to the message to send. **NULL** pointer must not be sent.

RETURN VALUE

OS_NO_ERR - The call was successful and the message was sent.
OS_Q_FULL - The queue cannot accept any more messages because it is full.
OS_ERR_EVENT_TYPE - A pointer to a queue was not passed.
OS_ERR_PEVENT_NULL - If **pevent** is a **NULL** pointer.
OS_ERR_POST_NULL_PTR - Attempting to post to a non mailbox.

LIBRARY

UCOS2.LIB

SEE ALSO

OSQAccept, OSQCreate, OSQFlush, OSQPend, OSQPost, OSQQuery

OSQQuery

```
INT8U OSQQuery (OS_EVENT *pevent, OS_Q_DATA *pdata);
```

DESCRIPTION

Obtains information about a message queue.

PARAMETERS

pevent	Pointer to message queue's event control block.
pdata	Pointer to a data structure for message queue information.

RETURN VALUE

OS_NO_ERR - The call was successful and the message was sent.
OS_ERR_EVENT_TYPE - Attempting to obtain data from a non queue.
OS_ERR_PEVENT_NULL - If pevent is a **NULL** pointer.

LIBRARY

UCOS2.LIB

SEE ALSO

OSQAccept, OSQCreate, OSQFlush, OSQPend, OSQPost, OSQPostFront

OSSchedLock

```
void OSSchedLock(void);
```

DESCRIPTION

Prevents task rescheduling. This allows an application to prevent context switches until it is ready for them. There must be a matched call to **OSSchedUnlock()** for every call to **OSSchedLock()**.

LIBRARY

UCOS2.LIB

SEE ALSO

OSSchedUnlock

OSSchedUnlock

```
void OSSchedUnlock(void);
```

DESCRIPTION

Allow task rescheduling. There must be a matched call to **OSSchedUnlock()** for every call to **OSSchedLock()**.

LIBRARY

UCOS2.LIB

SEE ALSO

OSSchedLock

OSSemAccept

```
INT16U OSSemAccept (OS_EVENT *pevent);
```

DESCRIPTION

This function checks the semaphore to see if a resource is available or if an event occurred. Unlike **OSSemPend()**, **OSSemAccept()** does not suspend the calling task if the resource is not available or the event did not occur.

PARAMETERS

pevent	Pointer to the desired semaphore's event control block

RETURN VALUE

Semaphore value:
If **>0**, semaphore value is decremented; value is returned before the decrement.
If **0**, then either resource is unavailable, event did not occur, or **NULL** or invalid pointer was passed to the function.

LIBRARY

UCOS2.LIB

SEE ALSO

OSSemCreate, OSSemPend, OSSemPost, OSSemQuery

OSSemCreate

```
OS_EVENT *OSSemCreate (INT16U cnt);
```

DESCRIPTION

Creates a semaphore.

PARAMETERS

cnt	The initial value of the semaphore.

RETURN VALUE

Pointer to the event control block (**OS_EVENT**) associated with the created semaphore, or **NULL** if no event control block is available.

LIBRARY

UCOS2.LIB

SEE ALSO

OSSemAccept, OSSemPend, OSSemPost, OSSemQuery

OSSemPend

```
void OSSemPend (OS_EVENT *pevent, INT16U timeout, INT8U *err);
```

DESCRIPTION

Waits on a semaphore.

PARAMETERS

pevent	Pointer to the desired semaphore's event control block
timeout	Time in clock ticks to wait for the resource. If 0, the task will wait until the resource becomes available or the event occurs.
err	Pointer to error message.

LIBRARY

UCOS2.LIB

SEE ALSO

OSSemAccept, OSSemCreate, OSSemPost, OSSemQuery

OSSemPost

```
INT8U OSSemPost (OS_EVENT *pevent);
```

DESCRIPTION

This function signals a semaphore.

PARAMETERS

pevent	Pointer to the desired semaphore's event control block

RETURN VALUE

OS_NO_ERR - The call was successful and the semaphore was signaled.
OS_SEM_OVF - If the semaphore count exceeded its limit. In other words, you have signalled the semaphore more often than you waited on it with either **OSSemAccept()** or **OSSemPend()**.
OS_ERR_EVENT_TYPE - If a pointer to a semaphore not passed.
OS_ERR_PEVENT_NULL - If **pevent** is a **NULL** pointer.

LIBRARY

UCOS2.LIB

SEE ALSO

OSSemAccept, OSSemCreate, OSSemPend, OSSemQuery

OSSemQuery

```
INT8U OSSemQuery (OS_EVENT *pevent, OS_SEM_DATA *pdata);
```

DESCRIPTION

Obtains information about a semaphore.

PARAMETERS

pevent	Pointer to the desired semaphore's event control block
pdata	Pointer to a data structure that will hold information about the semaphore.

RETURN VALUE

OS_NO_ERR - The call was successful and the message was sent.
OS_ERR_EVENT_TYPE - Attempting to obtain data from a non semaphore.
OS_ERR_PEVENT_NULL - If pevent is a **NULL** pointer.

LIBRARY

UCOS2.LIB

SEE ALSO

OSSemAccept, OSSemCreate, OSSemPend, OSSemPost

OSSetTickPerSec

```
INT16U OSSetTickPerSec(INT16U TicksPerSec);
```

DESCRIPTION

Sets the amount of ticks per second (from 1 - 2048). Ticks per second defaults to 64. If this function is used, the **#define OS_TICKS_PER_SEC** needs to be changed so that the time delay functions work correctly. Since this function uses integer division, the actual ticks per second may be slightly different that the desired ticks per second.

PARAMETERS

TicksPerSec Unsigned 16-bit integer.

RETURN VALUE

The actual ticks per second set, as an unsigned 16-bit integer.

LIBRARY

UCOS2.LIB

SEE ALSO

OSStart

OSStart

```
void OSStart(void);
```

DESCRIPTION

Starts the multitasking process, allowing µC/OS-II to manage the tasks that have been created. Before **OSStart()** is called, **OSInit()** MUST have been called and at least one task MUST have been created. This function calls **OSStartHighRdy** which calls **OSTaskSwHook** and sets **OSRunning** to TRUE.

LIBRARY

UCOS2.LIB

SEE ALSO

OSTaskCreate, OSTaskCreateExt

OSStatInit

```
void OSStatInit(void);
```

DESCRIPTION

Determines CPU usage.

LIBRARY

UCOS2.LIB

OSTaskChangePrio

```
INT8U OSTaskChangePrio (INT8U oldprio, INT8U newprio);
```

DESCRIPTION

Allows a task's priority to be changed dynamically. Note that the new priority MUST be available.

PARAMETERS

oldprio The priority level to change from.

newprio The priority level to change to.

RETURN VALUE

OS_NO_ERR - The call was successful.
OS_PRIO_INVALID - The priority specified is higher that the maximum allowed (i.e. ≥ **OS_LOWEST_PRIO**).
OS_PRIO_EXIST - The new priority already exist.
OS_PRIO_ERR - There is no task with the specified OLD priority (i.e. the OLD task does not exist).

LIBRARY

UCOS2.LIB

OSTaskCreate

INT8U OSTaskCreate(void (*task)(), void *pdata, INT16U stk_size, INT8U prio);

DESCRIPTION

Creates a task to be managed by µC/OS-II. Tasks can either be created prior to the start of multitasking or by a running task. A task cannot be created by an ISR.

PARAMETERS

task	Pointer to the task's starting address.
pdata	Pointer to a task's initial parameters.
stk_size	Number of bytes of the stack.
prior	The task's unique priority number.

RETURN VALUE

OS_NO_ERR - The call was successful.
OS_PRIO_EXIT - Task priority already exists (each task MUST have a unique priority).
OS_PRIO_INVALID - The priority specified is higher than the maximum allowed (i.e. ≥ **OS_LOWEST_PRIO**).

LIBRARY

UCOS2.LIB

SEE ALSO

OSTaskCreateExt

OSTaskCreateExt

```
INT8U OSTaskCreateExt (void   (*task)(), void *pdata, INT8U
   prio, INT16U id, INT16U stk_size, void *pext, INT16U opt);
```

DESCRIPTION

Creates a task to be managed by µC/OS-II. Tasks can either be created prior to the start of multitasking or by a running task. A task cannot be created by an ISR. This function is similar to **OSTaskCreate()** except that it allows additional information about a task to be specified.

PARAMETERS

task	Pointer to task's code.
pdata	Pointer to optional data area; used to pass parameters to the task at start of execution.
prio	The task's unique priority number; the lower the number the higher the priority.
id	The task's identification number (0...65535).
stk_size	Size of the stack in number of elements. If **OS_STK** is set to **INT8U**, **stk_size** corresponds to the number of bytes available. If **OS_STK** is set to **INT16U**, **stk_size** contains the number of 16-bit entries available. Finally, if **OS_STK** is set to **INT32U**, **stk_size** contains the number of 32-bit entries available on the stack.
pext	Pointer to a user-supplied Task Control Block (TCB) extension.
opt	The lower 8 bits are reserved by µC/OS-II. The upper 8 bits control application-specific options. Select an option by setting the corresponding bit(s).

RETURN VALUE

OS_NO_ERR - The call was successful.
OS_PRIO_EXIT - Task priority already exists (each task MUST have a unique priority).
OS_PRIO_INVALID - The priority specified is higher than the maximum allowed (i.e. ≥**OS_LOWEST_PRIO**).

LIBRARY

UCOS2.LIB

SEE ALSO

OSTaskCreate

OSTaskCreateHook

```
void OSTaskCreateHook(OS_TCB *ptcb);
```

DESCRIPTION

Called by µC/OS-II whenever a task is created. This call-back function resides in **UCOS2.LIB** and extends functionality during task creation by allowing additional information to be passed to the kernel, anything associated with a task. This function can also be used to trigger other hardware, such as an oscilloscope. Interrupts are disabled during this call, therefore, it is recommended that code be kept to a minimum.

PARAMETERS

ptcb Pointer to the TCB of the task being created.

LIBRARY

UCOS2.LIB

SEE ALSO

OSTaskDelHook

OSTaskDel

```
INT8U OSTaskDel (INT8U prio);
```

DESCRIPTION

Deletes a task. The calling task can delete itself by passing either its own priority number or **OS_PRIO_SELF** if it doesn't know its priority number. The deleted task is returned to the dormant state and can be re-activated by creating the deleted task again.

PARAMETERS

prio Task's priority number.

RETURN VALUE

OS_NO_ERR - The call was successful.
OS_TASK_DEL_IDLE - Attempting to delete uC/OS-II's idle task.
OS_PRIO_INVALID - The priority specified is higher than the maximum allowed (i.e. ≥ **OS_LOWEST_PRIO**) or, **OS_PRIO_SELF** not specified.
OS_TASK_DEL_ERR - The task to delete does not exist.
OS_TASK_DEL_ISR - Attempting to delete a task from an ISR.

LIBRARY

UCOS2.LIB

SEE ALSO

OSTaskDelReq

OSTaskDelHook

```
void OSTaskDelHook(OS_TCB *ptcb);
```

DESCRIPTION

Called by µC/OS-II whenever a task is deleted. This call-back function resides in **UCOS2.LIB.** Interrupts are disabled during this call, therefore, it is recommended that code be kept to a minimum.

PARAMETERS

ptcb Pointer to TCB of task being deleted.

LIBRARY

UCOS2.LIB

SEE ALSO

OSTaskCreateHook

OSTaskDelReq

```
INT8U OSTaskDelReq (INT8U prio);
```

DESCRIPTION

Notifies a task to delete itself. A well-behaved task is deleted when it regains control of the CPU by calling **OSTaskDelReq (OSTaskDelReq)** and monitoring the return value.

PARAMETERS

prio The priority of the task that is being asked to delete itself. **OS_PRIO_SELF** is used when asking whether another task wants the current task to be deleted.

RETURN VALUE

OS_NO_ERR - The task exists and the request has been registered.
OS_TASK_NOT_EXIST - The task has been deleted. This allows the caller to know whether the request has been executed.
OS_TASK_DEL_IDLE - If requesting to delete uC/OS-II's idletask.
OS_PRIO_INVALID - The priority specified is higher than the maximum allowed (i.e. ≥ **OS_LOWEST_PRIO**) or, **OS_PRIO_SELF** is not specified.
OS_TASK_DEL_REQ - A task (possibly another task) requested that the running task be deleted.

LIBRARY

UCOS2.LIB

SEE ALSO

OSTaskDel

OSTaskQuery

```
INT8U OSTaskQuery (INT8U prio, OS_TCB *pdata);
```

DESCRIPTION

Obtains a copy of the requested task's TCB.

PARAMETERS

prio	Priority number of the task.
pdata	Pointer to task's TCB.

RETURN VALUE

OS_NO_ERR - The requested task is suspended.
OS_PRIO_INVALID - The priority you specify is higher than the maximum allowed (i.e. ≥ **OS_LOWEST_PRIO**) or, **OS_PRIO_SELF** is not specified.
OS_PRIO_ERR - The desired task has not been created.

LIBRARY

UCOS2.LIB

OSTaskResume

```
INT8U OSTaskResume (INT8U prio);
```

DESCRIPTION

Resumes a suspended task. This is the only call that will remove an explicit task suspension.

PARAMETERS

prio The priority of the task to resume.

RETURN VALUE

OS_NO_ERR - The requested task is resumed.
OS_PRIO_INVALID - The priority specified is higher than the maximum allowed (i.e. ≥ **OS_LOWEST_PRIO**).
OS_TASK_NOT_SUSPENDED - The task to resume has not been suspended.

LIBRARY

UCOS2.LIB

SEE ALSO

OSTaskSuspend

OSTaskStatHook

```
void OSTaskStatHook();
```

DESCRIPTION

Called every second by µC/OS-II's statistics task. This function resides in **UCOS2.LIB** and allows an application to add functionality to the statistics task.

LIBRARY

UCOS2.LIB

OSTaskStkChk

```
INT8U OSTaskStkChk (INT8U prio, OS_STK_DATA *pdata);
```

DESCRIPTION

Check the amount of free memory on the stack of the specified task.

PARAMETERS

prio	The task's priority.
pdata	Pointer to a data structure of type **OS_STK_DATA.**

RETURN VALUE

OS_NO_ERR - The call was successful.
OS_PRIO_INVALID - The priority you specify is higher than the maximum allowed (i.e. > **OS_LOWEST_PRIO**) or, **OS_PRIO_SELF** not specified.
OS_TASK_NOT_EXIST - The desired task has not been created.
OS_TASK_OPT_ERR - If **OS_TASK_OPT_STK_CHK** was NOT specified when the task was created.

LIBRARY

UCOS2.LIB

SEE ALSO

OSTaskCreateExt

OSTaskSuspend

```
INT8U OSTaskSuspend (INT8U prio);
```

DESCRIPTION

Suspends a task. The task can be the calling task if the priority passed to **OSTaskSuspend()** is the priority of the calling task or **OS_PRIO_SELF**. This function should be used with great care. If a task is suspended that is waiting for an event (i.e. a message, a semaphore, a queue ...) the task will be prevented from running when the event arrives.

PARAMETERS

prio The priority of the task to suspend.

RETURN VALUE

OS_NO_ERR - The requested task is suspended.
OS_TASK_SUS_IDLE - Attempting to suspend the idle task (not allowed).
OS_PRIO_INVALID - The priority specified is higher than the maximum allowed (i.e. ≥ **OS_LOWEST_PRIO**) or, **OS_PRIO_SELF** is not specified .
OS_TASK_SUS_PRIO - The task to suspend does not exist.

LIBRARY

UCOS2.LIB

SEE ALSO

OSTaskResume

OSTaskSwHook

```
void OSTaskSwHook();
```

DESCRIPTION

Called whenever a context switch happens. The TCB for the task that is ready to run is accessed via the global variable **OSTCBHighRdy**, and the TCB for the task that is being switched out is accessed via the global variable **OSTCBCur**.

LIBRARY

UCOS2.LIB

OSTimeDly

```
void OSTimeDly (INT16U ticks);
```

DESCRIPTION

Delays execution of the task for the specified number of clock ticks. No delay will result if **ticks** is 0. If **ticks** is >0, then a context switch will result.

PARAMETERS

ticks	Number of clock ticks to delay the task.

LIBRARY

UCOS2.LIB

SEE ALSO

OSTimeDlyHMSM, OSTimeDlyResume, OSTimeDlySec

OSTimeDlyHMSM

```
INT8U OSTimeDlyHMSM (INT8U hours, INT8U minutes, INT8U seconds,
   INT16U milli);
```

DESCRIPTION

Delays execution of the task until specified amount of time expires. This call allows the delay to be specified in hours, minutes, seconds and milliseconds instead of ticks. The resolution on the milliseconds depends on the tick rate. For example, a 10 ms delay is not possible if the ticker interrupts every 100 ms. In this case, the delay would be set to 0. The actual delay is rounded to the nearest tick.

PARAMETERS

hours	Number of hours that the task will be delayed (max. is 255)
minutes	Number of minutes (max. 59)
seconds	Number of seconds (max. 59)
milli	Number of milliseconds (max. 999)

RETURN VALUE

OS_NO_ERR
OS_TIME_INVALID_MINUTES
OS_TIME_INVALID_SECONDS
OS_TIME_INVALID_MS
OS_TIME_ZERO_DLY

LIBRARY

UCOS2.LIB

SEE ALSO

OSTimeDly, OSTimeDlyResume, OSTimeDlySec

OSTimeDlyResume

```
INT8U OSTimeDlyResume (INT8U prio);
```

DESCRIPTION

Resumes a task that has been delayed through a call to either **OSTimeDly()** or **OSTimeDlyHMSM()**. Note that this function MUST NOT be called to resume a task that is waiting for an event with timeout. This situation would make the task look like a timeout occurred (unless this is the desired effect). Also, a task cannot be resumed that has called **OSTimeDlyHMSM()** with a combined time that exceeds 65535 clock ticks. In other words, if the clock tick runs at 100 Hz then, a delayed task will not be able to be resumed that called **OSTimeDlyHMSM(0, 10, 55, 350)** or higher.

PARAMETERS

prio Priority of the task to resume.

RETURN VALUE

OS_NO_ERR - Task has been resumed.
OS_PRIO_INVALID - The priority you specify is higher than the maximum allowed (i.e. ≥ **OS_LOWEST_PRIO**).
OS_TIME_NOT_DLY - Task is not waiting for time to expire.
OS_TASK_NOT_EXIST - The desired task has not been created.

LIBRARY

UCOS2.LIB

SEE ALSO

OSTimeDly, OSTimeDlyHMSM, OSTimeDlySec

OSTimeDlySec

```
INT8U OSTimeDlySec (INT16U seconds);
```

DESCRIPTION

Delays execution of the task until **seconds** expires. This is a low-overhead version of **OSTimeDlyHMSM** for seconds only.

PARAMETERS

seconds	The number of seconds to delay.

RETURN VALUE

OS_NO_ERR - The call was successful.
OS_TIME_ZERO_DLY - A delay of zero seconds was requested.

LIBRARY

UCOS2.LIB

SEE ALSO

OSTimeDly, OSTimeDlyHMSM, OSTimeDlyResume

OSTimeGet

```
INT32U OSTimeGet (void);
```

DESCRIPTION

Obtain the current value of the 32-bit counter that keeps track of the number of clock ticks.

RETURN VALUE

The current value of **OSTime**.

LIBRARY

UCOS2.LIB

SEE ALSO

OSTimeSet

OSTimeSet

```
void OSTimeSet (INT32U ticks);
```

DESCRIPTION

Sets the 32-bit counter that keeps track of the number of clock ticks.

PARAMETERS

ticks The value to set **OSTime** to.

LIBRARY

UCOS2.LIB

SEE ALSO

OSTimeGet

OSTimeTickHook

```
void OSTimeTickHook();
```

DESCRIPTION

This function, as included with Dynamic C, is a stub that does nothing except return. It is called every clock tick. If the user chooses to rewrite this function, code should be kept to a minimum as it will directly affect interrupt latency. This function must preserve any registers it uses, other than the ones that are preserved prior to the call to **OSTimeTickHook** at the beginning of the periodic interrupt (**periodic_isr** in **VDRIVER.LIB**). Therefore, **OSTimeTickHook** should be written in assembly. The registers saved by **periodic_isr** are: AF,IP, HL,DE and IX.

LIBRARY

UCOS2.LIB

OSVersion

```
INT16U OSVersion (void);
```

DESCRIPTION

Returns the version number of µC/OS-II. The returned value corresponds to µC/OS-II's version number multiplied by 100; i.e., version 2.00 would be returned as 200.

RETURN VALUE

Version number multiplied by 100.

LIBRARY

UCOS2.LIB

outchrs

```
char outchrs(char c, int n, int (*putc) () );
```

DESCRIPTION

Use **putc** to output **n** times the character **c**.

PARAMETERS

c	Character to output
n	Number of times to output
putc	Routine to output one character. The function pointed to by **putc** should take a character argument.

RETURN VALUE

The character in parameter **c**.

LIBRARY

STDIO.LIB

SEE ALSO

outstr

outstr

```
char *outstr(char *string, int (*putc)() );
```

DESCRIPTION

Output the string pointed to by **string** via calls to **putc**. **putc** should take a one-character parameter.

PARAMETERS

string	String to output
putc	Routine to output one character. The function pointed to by **putc** should take a character argument.

RETURN VALUE

Pointer to **NULL** at end of string.

LIBRARY

STDIO.LIB

SEE ALSO

outchrs

paddr

```
unsigned long paddr(void* pointer);
```

DESCRIPTION

Converts a logical pointer into its physical address. Use caution when converting address in the E000-FFFF range. Returns the address based on the XPC on entry.

PARAMETERS

pointer	The pointer to convert.

RETURN VALUE

The physical address of the pointer.

LIBRARY

XMEM.LIB

pktXclose

```
void pktXclose();
```

DESCRIPTION

Disables serial port X, where X is A|B|C|D. Starting with Dynamic C version 7.25, the functions **pktEclose()** and **pktFclose()** may be used with the Rabbit 3000 microprocessor.

LIBRARY

PACKET.LIB

pktXgetErrors

```
char pktXgetErrors();
```

DESCRIPTION

Gets a bit field with flags set for any errors that occured on port X, where X is A|B|C|D. These flags are then cleared, so that a particular error will only cause the flag to be set once.

Starting with Dynamic C version 7.25, the functions **pktEgetErrors()** and **pktFgetErrors()** may be used with the Rabbit 3000 microprocessor.

RETURN VALUE

A bit field with flags for various errors. The errors along with their bit masks are as follows:

PKT_BUFFEROVERFLOW	0x01
PKT_RXOVERRUN	0x02
PKT_PARITYERROR	0x04
PKT_NOBUFFER	0x08

LIBRARY

PACKET.LIB

pktXinitBuffers

```
int pktXinitBuffers(int buf_count, int buf_size); X = A|B|C|D
```

DESCRIPTION

Allocates extended memory for channel X receive buffers. This function should not be called more than once in a program. The total memory allocated is **buf_count***(**buf_size** + 2) bytes.

Starting with Dynamic C version 7.25, the functions **pktEinitBuffers()** and **pktFinitBuffers()** may be used with the Rabbit 3000 microprocessor.

PARAMETERS

buf_count The number of buffers to allocate. Each buffer can store one received packet. Increasing this number allows for more pending packets and a larger latency time before packets must be processed by the user's program.

buf_size The number of bytes each buffer can accomodate. This should be set to the size of the largest possible packet that can be expected.

RETURN VALUE

1: Success, extended memory was allocated.
0: Failure, no memory allocated, the packet channel cannot be used.

LIBRARY

PACKET.LIB

pktXopen

```
int pktXopen(long baud, int mode, char options,
   int (*test_packet)());
```

DESCRIPTION

Opens serial port X, where X is A|B|C|D. Starting with Dynamic C version 7.25, the functions **pktEopen()** and **pktFopen()** may be used with the Rabbit 3000 microprocessor.

PARAMETERS

baud Bits per second of data transfer: minimum is 2400.

mode Type of packet scheme used, the options are:

- **PKT_GAPMODE**
- **PKT_9BITMODE**
- **PKT_CHARMODE**

options Further specification for the packet scheme. The value of this depends on the mode used:

- gap mode - minimum gap size(in byte times)
- 9-bit mode - type of 9-bit protocol
  - **PKT_RABBITSTARTBYTE**
  - **PKT_LOWSTARTBYTE**
  - **PKT_HIGHSTARTBYTE**
- char mode - character marking start of packet

test_packet Pointer to a function that tests for completeness of a packet. The function should return 1 if the packet is complete, or 0 if more data should be read in. For gap mode the test function is not used and should be set to **NULL**.

RETURN VALUE

1: The baud set on the rabbit is the same as the input baud.
0: The baud set on the rabbit does not match the input baud.

LIBRARY

PACKET.LIB

pktXreceive

```
int pktXreceive(void *buffer, int buffer_size);
```

DESCRIPTION

Gets a received packet, if there is one, from serial port X, where X is A|B|C|D. Starting with Dynamic C version 7.25, the functions **pktEreceive()** and **pktFreceive()** may be used with the Rabbit 3000 microprocessor.

PARAMETERS

buffer A buffer for the packet to be written into.

buffer_size Length of the data buffer.

RETURN VALUE

>0: Number of bytes in the successfully received packet.
0: No new packet has been received.
-1: The packet is too large for the given buffer
-2: A needed **test_packet** function is not defined

LIBRARY

PACKET.LIB

pktXsend

```
int pktXsend(void *send_buffer, int buffer_length, char delay);
```

DESCRIPTION

Initiates the sending of a packet of data using serial port X, where X is A|B|C|D. This function will always return immediately. If there is already a packet being transmitted, this call will return 0 and the packet will not be transmitted, otherwise it will return 1.

pktXsending() checks if the packet is done transmitting. The system will be using the buffer until then.

Starting with Dynamic C version 7.25, the functions **pktEsend()** and **pktFsend()** may be used with the Rabbit 3000 microprocessor.

PARAMETERS

send_buffer	The data to be sent
buffer_length	Length of the data buffer to transmit
delay	The number of byte times to delay before sending the data (0-255) This is used to implement protocol-specific delays between packets

RETURN VALUE

1: The packet is going to be transmitted.
0: There is already a packet transmitting, and the new packet was refused.

LIBRARY

PACKET.LIB

pktXsending

```
int pktXsending();
```

DESCRIPTION

Tests if a packet is currently being sent on serial port X, where X=A|B|C|D. If **pktXsending()** returns true, the transmitter is busy and cannot accept another packet.

Starting with Dynamic C version 7.25, the functions **pktEsending()** and **pktFsending()** may be used with the Rabbit 3000 microprocessor.

RETURN VALUE

1: A packet is being transmitted.
0: Port X is idle, ready for a new packet.

LIBRARY

PACKET.LIB

pktXsetParity

```
void pktXsetParity(char mode);
```

DESCRIPTION

Configures parity generation and checking. Can also configure for 2 stop bits.

Starting with Dynamic C version 7.25, the functions **pktEsetParity()** and **pktFsetParity()** may be used with the Rabbit 3000 microprocessor.

PARAMETERS

mode Code for mode of parity bit:

- **PKT_NOPARITY** - no parity bit (8N1 format, default)
- **PKT_OPARITY** - odd parity (8O1 format)
- **PKT_EPARITY** - even parity (8E1 format)
- **PKT_TWOSTOP** - an extra stop bit (8N2 format)

LIBRARY

PACKET.LIB

poly

```
float poly(float x, int n, float c[]);
```

DESCRIPTION

Computes polynomial value by Horner's method. For example, for the fourth-order polynomial $10x^4 - 3x^2 + 4x + 6$, **n** would be 4 and the coefficients would be

```
c[4] = 10.0
c[3] =  0.0
c[2] = -3.0
c[1] =  4.0
c[0] =  6.0
```

PARAMETERS

x Variable of the polynomial.

n The order of the polynomial

c Array containing the coefficients of each power of **x**.

RETURN VALUE

The polynomial value.

LIBRARY

MATH.LIB

pow

```
float pow(float x, float y);
```

DESCRIPTION

Raises **x** to the **y**th power.

PARAMETERS

x	Value to be raised
y	Exponent

RETURN VALUE

x to the **y**th power

LIBRARY

MATH.LIB

SEE ALSO

exp, pow10, sqrt

pow10

```
float pow10(float x);
```

DESCRIPTION

10 to the power of **x**.

PARAMETERS

x	Exponent

RETURN VALUE

10 raised to power **x**

LIBRARY

MATH.LIB

SEE ALSO

pow, exp, sqrt

powerspectrum

```
void powerspectrum(int *x, int N, *int blockexp);
```

DESCRIPTION

Computes the power spectrum from a complex spectrum according to

$$Power[k] = (\text{Re } X[k])^2 + (\text{Im } X[k])^2$$

The **N**-point power spectrum replaces the **N**-point complex spectrum. The power of each complex spectral component is computed as a 32-bit fraction. Its more significant 16-bits replace the imaginary part of the component; its less significant 16-bits replace the real part.

If the complex input spectrum is a positive-frequency spectrum computed by **fftreal(),** the imaginary part of the $X[0]$ term (stored **x[1]**) will contain the real part of the *fmax* term and will affect the calculation of the dc power. If the dc power or the *fmax* power is important, the *fmax* term should be retrieved from **x[1]** and **x[1]** set to zero before calling **powerspectrum()**.

The power of the *k* th term can be retrieved via
P[k]=*(long*)&x[2k]*2^blockexp.

The value of **blockexp** is first doubled to reflect the squaring operation applied to all elements in array **x**. Then it is further increased by 1 to reflect an inherent division-by-two that occurs during the squaring operation.

PARAMETERS

x	pointer to **N**-element array of complex fractions.
N	number of complex elements in array **x**.
blockexp	pointer to integer block exponent.

LIBRARY

FFT.LIB

SEE ALSO

fftcplx, fftcplxinv, fftreal, fftrealinv, hanncplx, hannreal

premain

```
void premain();
```

DESCRIPTION

Dynamic C calls **premain** to start initialization functions such as **VdInit**. The final thing **premain** does is call **main**. This function should never be called by an application program. It is included here for informational purposes only.

LIBRARY

PROGRAM.LIB

printf

```
int printf(char *fmt, ...);
```

DESCRIPTION

This function is similar to **sprintf()**, but outputs the formatted string to the Stdio window in Dynamic C. Please refer to the description of **sprintf()** for more details.

This function is task reentrant.

Prior to Dynamic C 7.25, it would work only with the controller in program mode connected to the PC running Dynamic C. As of Dynamic C 7.25, it is possible to redirect printf output to a serial port during run mode by defining a macro to specify the serial port. See the sample program **/SAMPLES/STDIO_SERIAL.C** for more information.

PARAMETERS

format	String to be formatted.
...	Format arguments.

RETURN VALUE

Number of characters written

LIBRARY

STDIO.LIB

SEE ALSO

sprintf

putchar

```
void putchar(int ch);
```

DESCRIPTION

Puts a single character to Stdout. The user should make sure only one process calls this function at a time.

PARAMETERS

ch Character to be displayed.

LIBRARY

STDIO.LIB

SEE ALSO

puts, getchar

puts

```
int puts(char *s);
```

DESCRIPTION

This function displays the string on the stdio window in Dynamic C. The Stdio window is responsible for interpreting any escape code sequences contained in the string. Only one process at a time should call this function.

PARAMETERS

s Pointer to string argument to be displayed.

RETURN VALUE

1: Success.

LIBRARY

STDIO.LIB

SEE ALSO

putchar, gets

pwm_init

```
unsigned long pwm_init(unsigned long frequency);
```

DESCRIPTION

Sets the base frequency for the pulse width modulation (PWM) and enables the PWM driver on all four channels. The base frequency is the frequency without pulse spreading. Pulse spreading (see **pwm_set()**) will increase the frequency by a factor of 4.

This function is intended for use with the Rabbit 3000 microprocessor.

PARAMETER

frequency Requested frequency (in Hz)

RETURN VALUE

The actual frequency that was set. This will be the closest possible match to the requested frequency.

LIBRARY

R3000.LIB

pwm_set

```
int pwm_set(int channel, int duty_cycle, int options);
```

DESCRIPTION

Sets a duty cycle for one of the pulse width modulation (PWM) channels. The duty cycle can be a value from 0 to 1024, where 0 is logic low the whole time, and 1024 is logic high the whole time. Option flags are used to enable features on an individual PWM channel. Bit masks for these are:

- **PWM_SPREAD** - sets pulse spreading. The duty cycle is spread over four seperate pulses to increase the pulse frequency.
- **PWM_OPENDRAIN** - sets the PWM output pin to be open-drain instead of a normal push-pull logic output.

This function is intended for use with the Rabbit 3000 microprocessor.

PARAMETERS

channel	PWM channel(0 to 3)
duty_cycle	value from 0 to 1024
options	combination of optional flags(see above)

RETURN VALUE

0: Success.
-1: Error, an invalid channel number is used.
-2: Error, requested **duty_cycle** is invalid.

LIBRARY

R3000.LIB

qd_error

```
char qd_error(int channel);
```

DESCRIPTION

Gets the current error bits for that qd channel. This function is intended to be used with the Rabbit 3000 microprocessor.

PARAMETERS

channel The channel to read errors from (currently 1 or 2).

RETURN VALUE

Set of error flags, that can be decoded with the following masks:

- **QD_OVERFLOW** 0x01
- **QD_UNDERFLOW** 0x02

LIBRARY

R3000.LIB

qd_init

```
void qd_init(int iplevel);
```

DESCRIPTION

Initializes the quadrature decoders and sets up the ISR. This must be called before any other QD functions are used. Sets up the lower nibble of port F to be the QD input pins.

This function is intended for use with the Rabbit 3000 microprocessor.

PARAMETERS

iplevel The interrupt priority for the ISR that handles the count overflow. This should usually be 1.

LIBRARY

R3000.LIB

qd_read

```
long qd_read(int channel);
```

DESCRIPTION

Reads the current quadrature decoder count. Since this function waits for a clear reading, it can potentially block if there is enough flutter in the decoder count.

This function is intended to be used with the Rabbit 3000 microprocessor.

PARAMETERS

channel The channel to read (currently 1 or 2).

RETURN VALUE

Returns a signed long for the current count.

LIBRARY

R3000.LIB

qd_zero

```
void qd_zero(int channel);
```

DESCRIPTION

Sets the count for a channel to 0. This function is intended to be used with the Rabbit 3000 microprocessor.

PARAMETERS

channel The channel to reset (currently 1 or 2)

LIBRARY

R3000.LIB

qsort

```
int qsort(char *base, unsigned n, unsigned s, int (*cmp) ());
```

DESCRIPTION

Quick sort with center pivot, stack control, and easy-to-change comparison method. This version sorts fixed-length data items. It is ideal for integers, longs, floats and packed string data without delimiters.

Can sort raw integers, longs, floats or strings. However, the string sort is not efficient.

PARAMETERS

base	Base address of the raw string data
n	Number of blocks to sort
s	Number of bytes in each block
cmp	User-supplied compare routine for two block pointers, **p** and **q**, that returns an int with the same rules used by Unix **strcmp(p,q)**: = 0: Blocks **p** and **q** are equal < 0: **p** is less than **q** > 0: **p** is greater than **q** Beware of using ordinary **strcmp()**—it requires a **NULL** at the end of each string.

RETURN VALUE

0 if the operation is successful.

LIBRARY

SYS.LIB

EXAMPLE

```
// Sort an array of integers.
int mycmp(p,q) int *p,*q; { return (*p - *q);}
const int q[10] = {12,1,3,-2,16,7,9,34,-90,10};
const int p[10] = {12,1,3,-2,16,7,9,34,-90,10};
main() {
    int i;
    qsort(p,10,2,mycmp);
    for(i=0;i<10;++i) printf("%d. %d, %d\n",i,p[i],q[i]);
}
```

```
Output from the above sample program:
0. -90,  12
1.  -2,   1
2.   1,   3
3.   3,  -2
4.   7,  16
5.   9,   7
6.  10,   9
7.  12,  34
8.  16, -90
9.  34,  10
```

rad

```
float rad(float x);
```

DESCRIPTION

Convert degrees (360 for one rotation) to radians (2π for a rotation).

PARAMETERS

x Degree value to convert

RETURN VALUE

The radians equivalent of degree.

LIBRARY

SYS.LIB

SEE ALSO

deg

rand

```
float rand(void);
```

DESCRIPTION

Returns a uniformly distributed random number in the range 0.0 ≤v < 1.0. Uses algorithm:

$$\mathbf{rand = (5*rand) modulo\ 2^{32}}$$

A default seed value is set on startup, but can be changed with the **srand()** function. **rand()** is not reentrant.

RETURN VALUE

A uniformly distributed random number: 0.0 ≤ **v** < 1.0.

LIBRARY

MATH.LIB

SEE ALSO

randb, randg, srand

randb

```
float randb(void);
```

DESCRIPTION

Uses algorithm:

$$\mathbf{rand = (5*rand) modulo\ 2^{32}}$$

A default seed value is set on startup, but can be changed with the **srand()** function. **randb()** is not reentrant.

RETURN VALUE

Returns a uniformly distributed random number: -1.0 ≤ **v** < 1.0.

LIBRARY

MATH.LIB

SEE ALSO

rand, randg, srand

randg

```
float randg(void);
```

DESCRIPTION

Returns a gaussian-distributed random number in the range -16.0 ≤ v < 16.0 with a standard deviation of approximately 2.6. The distribution is made by adding 16 random numbers (see **rand()**). This function is not task reentrant.

RETURN VALUE

A gaussian distributed random number: -16.0 ≤ **v** <16.0.

LIBRARY

MATH.LIB

SEE ALSO

rand, randb, srand

RdPortE

```
int RdPortE(unsigned int port);
```

DESCRIPTION

Reads an external I/O register specified by the argument.

PARAMETERS

port Address of external parallel port data register.

RETURN VALUE

Returns an integer, the lower 8 bits of which contain the result of reading the port specified by the argument. Upper byte contains zero.

LIBRARY

SYSIO.LIB

SEE ALSO

RdPortI, BitRdPortI, WrPortI, BitWrPortI, BitRdPortE, WrPortE, BitWrPortE

RdPortI

```
int RdPortI(int port);
```

DESCRIPTION

Reads an internal I/O port specified by the argument.

PARAMETERS

port Address of internal parallel port data register.

RETURN VALUE

Returns an integer, the lower 8 bits of which contain the result of reading the port specified by the argument. Upper byte contains zero.

LIBRARY

SYSIO.LIB

SEE ALSO

RdPortE, BitRdPortI, WrPortI, BitWrPortI, BitRdPortE, WrPortE, BitWrPortE

read_rtc

```
unsigned long read_rtc(void);
```

DESCRIPTION

Reads the Real-time Clock (RTC) directly. Use with caution! In most cases use long variable **SEC_TIMER,** which contains the same result, unless the RTC has been changed since the start of the program. If you are running the processor off the 32 kHz crystal, use **read_rtc_32kHz()** instead.

RETURN VALUE

Time in seconds since January 1, 1980 (if RTC set correctly).

LIBRARY

RTCLOCK.LIB

SEE ALSO

write_rtc

read_rtc_32kHz

```
unsigned long read_rtc_32kHz(void);
```

DESCRIPTION

Reads the real-time clock directly when the Rabbit processor is running off the 32 kHz oscillator. See **read_rtc** for more details.

RETURN VALUE

Time in seconds since January 1, 1980 (if RTC set correctly).

LIBRARY

RTCLOCK.LIB

readUserBlock

```
int readUserBlock(void *dest, int addr, int numbytes);
```

DESCRIPTION

Reads a number of bytes from the user block on the primary flash to a buffer in root memory. NOTE: Portions of the user block may be used by the BIOS for your board to store values. For example, any board with an A to D converter will require the BIOS to write calibration constants to the user block. For some versions of the BL2000 and the BL2100 this memory area is 0x1C00 to 0x1FFF. See the user's manual for your particular board for more information before overwriting any part of the user block. Also, see the *Rabbit 2000 Microprocessor Designer's Handbook* for more information on the user block.

PARAMETERS

dest	Pointer to destination to copy data to.
addr	Address offset in user block to read from.
numbytes	Number of bytes to copy.

RETURN VALUE

0: Success
-1: Invalid address or range

LIBRARY

IDBLOCK.LIB

SEE ALSO

writeUserBlock

res

```
void res(void *address, unsigned int bit);
```

DESCRIPTION

Dynamic C may expand this call inline

Clears specified bit at memory address to 0. bit may be from 0 to 31. This is equivalent to the following expression, but more efficient:

```
*(long *)address &= ~(1L << bit)
```

PARAMETERS

address	Address of byte containing bits 7-0
bit	Bit location where 0 represents the least significant bit

LIBRARY

UTIL.LIB

SEE ALSO

RES

RES

```
void RES(void *address, unsigned int bit);
```

DESCRIPTION

Dynamic C may expand this call inline.

Clears specified bit at memory address to 0. bit may be from 0 to 31. This is equivalent to the following expression, but more efficient:

```
*(long *)address &= ~(1L << bit)
```

PARAMETERS

address	Address of byte containing bits 7-0
bit	Bit location where 0 represents the least significant bit

LIBRARY

UTIL.LIB

SEE ALSO

res

ResetErrorLog

```
void ResetErrorLog();
```

DESCRIPTION

This function resets the exception and restart type counts in the error log buffer header. This function is not called by default from anywhere. It should be used to initialized the error log when a board is programmed by means other than Dynamic C, cloning, the Rabbit Field Utility (RFU), or a service processor. For example, if boards are mass produced with pre-programmed flash chips, then the test program that runs on the boards should call this function.

LIBRARY

ERRORS.LIB

root2xmem

```
int root2xmem(unsigned long dest, void *src, unsigned len);
```

DESCRIPTION

Stores **len** characters from logical address **src** to physical address **dest**.

PARAMETERS

dest	Physical address
src	Logical address
len	Numbers of bytes

RETURN VALUE

0: Success
-1: Attempt to write flash memory area, nothing written
-2: Source not all in root

LIBRARY

XMEM.LIB

SEE ALSO

xalloc, xmem2root

runwatch

```
void runwatch();
```

DESCRIPTION

Runs and updates watch expressions if Dynamic C has requested it with a **Ctrl-U**. Should be called periodically in user program.

LIBRARY

SYS.LIB

serCheckParity

```
int serCheckParity(char rx_byte, char parity);
```

DESCRIPTION

This function is different from the other serial routines in that it does not specify a particular serial port. This function takes any 8-bit character and tests it for correct parity. It will return true if the parity of **rx_byte** matches the parity specified. This function is useful for checking individual characters when using a 7-bit data protocol.

PARAMETERS

rx_byte The 8 bit character being tested for parity.

parity The character 'O' for odd parity, or the character 'E' for even parity.

RETURN VALUE

1: Parity of the byte being tested matches the parity supplied as an argument.
0: Parity of the byte does not match.

LIBRARY

RS232.LIB

serXclose

```
void serXclose(); /* where X = A|B|C|D */
```

DESCRIPTION

Disables serial port X. This function is non-reentrant.

Starting with Dynamic C version 7.25, the functions **serEclose()** and **serFclose()** may be used with the Rabbit 3000 microprocessor.

LIBRARY

RS232.LIB

serXdatabits

```
void serXdatabits(state); /* where X = A|B|C|D */
```

DESCRIPTION

Sets the number of data bits in the serial format for this channel. Currently seven or eight bit modes are supported. A call to **serXopen()** must be made before calling this function. This function is non-reentrant.

Starting with Dynamic C version 7.25, the functions **serEdatabits()** and **serFdatabits()** may be used with the Rabbit 3000 microprocessor.

PARAMETERS

state	An integer indicating what bit mode to use. It is best to use one of the macros provided for this:
PARAM_7BIT	Configures serial port to use seven bit data.
PARAM_8BIT	Configures serial port to use eight bit data (default).

LIBRARY

RS232.LIB

serXflowcontrolOff

```
void serXflowcontrolOff(); /* where X = A|B|C|D */
```

DESCRIPTION

Turns off hardware flow control for serial port X. A call to **serXopen()** must be made before calling this function. This function is non-reentrant.

Starting with Dynamic C version 7.25, the functions **serEflowcontrolOff()** and **serFflowcontrolOff()** may be used with the Rabbit 3000 microprocessor.

LIBRARY

RS232.LIB

serXflowcontrolOn

```
void serXflowcontrolOn(); /* where X = A|B|C|D */
```

DESCRIPTION

Turns on hardware flow control for channel X. This enables two digital lines that handle flow control, CTS (clear to send) and RTS (ready to send). CTS is an input that will be pulled active low by the other system when it is ready to receive data. The RTS signal is an output that the system uses to indicate that it is ready to receive data; it is driven low when data can be received. A call to **serXopen()** must be made before calling this function.

This function is non-reentrant.

Starting with Dynamic C version 7.25, the functions **serEflowcontrolOn()** and **serFflowcontrolOn()** may be used with the Rabbit 3000 microprocessor.

If pins for the flow control lines are not explicitly defined, defaults will be used and compiler warnings will be issued. The locations of the flow control lines are specified using a set of 5 macros.

SERX_RTS_PORT	Data register for the parallel port that the RTS line is on. e.g. PCDR
SERA_RTS_SHADOW	Shadow register for the RTS line's parallel port. e.g. PCDRShadow
SERA_RTS_BIT	The bit number for the RTS line
SERA_CTS_PORT	Data register for the parallel port that the CTS line is on
SERA_CTS_BIT	The bit number for the CTS line

LIBRARY

RS232.LIB

serXgetc

```
int serXgetc(); /* where X = A|B|C|D */
```

DESCRIPTION

Get next available character from serial port X read buffer. This function is non-reentrant.

Starting with Dynamic C version 7.25, the functions **serEgetc()** and **serFgetc()** may be used with the Rabbit 3000 microprocessor.

RETURN VALUE

Success: the next character in the low byte, **0** in the high byte
Failure: **-1**

LIBRARY

RS232.LIB

EXAMPLE

```
// echoes characters
main() {
    int c;
    serAopen(19200);
    while (1) {
        if ((c = serAgetc()) != -1) {
            serAputc(c);
        }
    }
    serAclose()
}
```

serXgetError

```
int serXgetError(); /* where X = A|B|C|D */
```

DESCRIPTION

Returns a byte of error flags, with bits set for any errors that occurred since the last time this function was called. Any bits set will be automatically cleared when this function is called, so a particular error will only be reported once. This function is non-reentrant.

The flags are checked with bitmasks to determine which errors occurred. Error bitmasks:

- **SER_PARITY_ERROR**
- **SER_OVERRUN_ERROR**

Starting with Dynamic C version 7.25, the functions **serEgetError()** and **serFgetError()** may be used with the Rabbit 3000 microprocessor.

RETURN VALUE

The error flags byte.

LIBRARY

RS232.LIB

serXopen

```
int serXopen(long baud); /* where X = A|B|C|D */
```

DESCRIPTION

Opens serial port X. This function is non-reentrant.

The user must define the buffer sizes for each port being used with the buffer size macros **XINBUFSIZE** and **XOUTBUFSIZE**. The values must be a power of 2 minus 1, e.g.

```
#define XINBUFSIZE   63
#define XOUTBUFSIZE  127
```

Defining the buffer sizes to 2^n - 1 makes the circular buffer operations very efficient. If a value not equal to 2^n- 1 is defined, a default of 31 is used and a compiler warning is given.

Starting with Dynamic C version 7.25, the functions **serEopen()** and **serFopen()** may be used with the Rabbit 3000 microprocessor.

Note: The alternate pins on parallel port D can be used for serial port B by defining **SERB_USEPORTD** at the beginning of a program. See the section on parallel port D in the Rabbit documentation for more detail on the alternate serial port pins.

PARAMETERS

baud	Bits per second of data transfer. Note that the baud rate must be greater than or equal to the peripheral clock frequency ÷ 8192.

RETURN VALUE

1: The baud rate achieved on the Rabbit is the same as the input baud rate. The software was able to calculate a valid divisor for the requested baud rate within 5%.

0: The baud rate achieved on the Rabbit does not match the input baud rate.

LIBRARY

RS232.LIB

SEE ALSO

serXgetc, serXpeek, serXputs, serXwrite, cof_serXgetc, cof_serXgets, cof_serXread, cof_serXputc, cof_serXputs, cof_serXwrite, serXclose

serXparity

```
void serXparity(int parity_mode); /* where X = A|B|C|D */
```

DESCRIPTION

Sets parity mode for channel X. A call to **serXopen()** must be made before calling this function.

Parity generation for 8 bit data can be unusually slow due to the current method for generating high 9th bits. Whenever, a 9th high bit is needed, the UART is disabled for approximately 10 baud times to create a long stop bit that should be recognized by the receiver as a high 9th bit.

The long delay is imposed because we are using the serial port itself to handle timing for the delay. Creating a shorter delay would the require use of some other timer resource.

This function is non-reentrant.

Starting with Dynamic C version 7.25, the functions **serEparity()** and **serFparity()** may be used with the Rabbit 3000 microprocessor.

PARAMETERS

parity_mode An integer indicating what parity mode to use. It is best to use one of the macros provided:

- **PARAM_NOPARITY** - Disables parity handling (default).
- **PARAM_OPARITY** - Configures serial port to check/generate for odd parity.
- **PARAM_EPARITY** - Configures serial port to check/generate for even parity.
- **PARAM_2STOP** - Configures serial port to generate 2 stop bits.

LIBRARY

RS232.LIB

serXpeek

```
int serXpeek(); /* where X = [A|B|C|D] */
```

DESCRIPTION

Returns 1st character in input buffer X, without removing it from the buffer. This function is non-reentrant.

Starting with Dynamic C version 7.25, the functions **serEpeek()** and **serFpeek()** may be used with the Rabbit 3000 microprocessor.

RETURN VALUE

An integer with 1st character in buffer in the low byte
-1 if the buffer is empty

LIBRARY

RS232.LIB

serXputc

```
int serXputc(char c); /* where X = A|B|C|D */
```

DESCRIPTION

Writes a character to serial port X write buffer. This function is non-reentrant.

Starting with Dynamic C version 7.25, the functions **serEputc()** and **serFputc()** may be used with the Rabbit 3000 microprocessor.

PARAMETERS

c Character to write to serial port X write buffer.

RETURN VALUE

0 if buffer locked or full, **1** if character sent.

LIBRARY

RS232.LIB

EXAMPLE

```
main() {    // echoes characters
    int c;
    serAopen(19200);
    while (1) {
        if ((c = serAgetc()) != -1) {
            serAputc(c);
        }
    }
    serAclose();
}
```

serXputs

```
int serXputs(char* s); /* where X = A|B|C|D */
```

DESCRIPTION

Calls **serXwrite(s, strlen(s))**. This function is non-reentrant.

Starting with Dynamic C version 7.25, the functions **serEputs()** and **serFputs()** may be used with the Rabbit 3000 microprocessor.

PARAMETERS

s **NULL**-terminated character string to write

RETURN VALUE

The number of characters actually sent from serial port X.

LIBRARY

RS232.LIB

EXAMPLE

```
// writes a null-terminated string of characters, repeatedly
main() {
    const char s[] = "Hello Z-World";
    serAopen(19200);
    while (1) {
        serAputs(s);
    }
    serAclose();
}
```

serXrdFlush

```
void serXrdFlush(); /* where X = A|B|C|D */
```

DESCRIPTION

Flushes serial port X input buffer. This function is non-reentrant.

Starting with Dynamic C version 7.25, the functions **serErdFlush()** and **serFrdFlush()** may be used with the Rabbit 3000 microprocessor.

LIBRARY

RS232.LIB

serXrdFree

```
int serXrdFree(); /* where X = A|B|C|D */
```

DESCRIPTION

Calculates the number of characters of unused data space. This function is non-reentrant.

Starting with Dynamic C version 7.25, the functions **serErdFree()** and **serFrdFree()** may be used with the Rabbit 3000 microprocessor.

RETURN VALUE

The number of chars it would take to fill input buffer X.

LIBRARY

RS232.LIB

serXrdUsed

```
int serXrdUsed(); /* where X = A|B|C|D */
```

DESCRIPTION

Calculates the number of characters ready to read from the serial port receive buffer. This function is non-reentrant.

Starting with Dynamic C version 7.25, the functions **serErdUsed()** and **serFrdUsed()** may be used with the Rabbit 3000 microprocessor.

RETURN VALUE

The number of characters currently in serial port X receive buffer.

LIBRARY

RS232.LIB

serXread

```
int serXread(void *data, int length, unsigned long tmout);
   /* where X = A|B|C|D */
```

DESCRIPTION

Reads **length** bytes from serial port X or until **tmout** milliseconds transpires between bytes. The countdown of **tmout** does not begin until a byte has been received. A timeout occurs immediately if there are no characters to read. This function is non-reentrant.

Starting with Dynamic C version 7.25, the functions **serEread()** and **serFread()** may be used with the Rabbit 3000 microprocessor.

PARAMETERS

data	Data structure to read from serial port X
length	Number of bytes to read
tmout	Maximum wait in milliseconds for any byte from previous one

RETURN VALUE

The number of bytes read from serial port X.

LIBRARY

RS232.LIB

EXAMPLE

```
// echoes a blocks of characters
main() {
    int n;
    char s[16];
    serAopen(19200);
    while (1) {
        if ((n = serAread(s, 15, 20)) > 0) {
            serAwrite(s, n);
        }
    }
    serAclose();
}
```

serXwrFlush

```
void serXwrFlush(); /* where X = A|B|C|D */
```

DESCRIPTION

Flushes serial port X transmit buffer. This function is non-reentrant.

Starting with Dynamic C version 7.25, the functions **serEwrFlush()** and **serFwrFlush()** may be used with the Rabbit 3000 microprocessor.

LIBRARY

RS232.LIB

serXwrFree

```
int serXwrfree(); /* where X = A|B|C|D */
```

DESCRIPTION

Calculates the free space in the serial port transmit buffer. This function is non-reentrant.

Starting with Dynamic C version 7.25, the functions **serEwrFree()** and **serFwrFree()** may be used with the Rabbit 3000 microprocessor.

RETURN VALUE

The number of characters the serial port transmit buffer can accept before becoming full.

LIBRARY

RS232.LIB

serXwrite

```
int serXwrite(void *data, int length); /* where X = A|B|C|D */
```

DESCRIPTION

Transmits **length** bytes to serial port X. This function is non-reentrant.

Starting with Dynamic C version 7.25, the functions **serEwrite()** and **serFwrite()** may be used with the Rabbit 3000 microprocessor.

PARAMETERS

data	Data structure to write to serial port X.
length	Number of bytes to write

RETURN VALUE

The number of bytes successfully written to the serial port.

LIBRARY

RS232.LIB

EXAMPLE

```
// writes a block of characters, repeatedly
main() {
    const char s[] = "Hello Z-World";
    serAopen(19200);
    while (1) {
        serAwrite(s, strlen(s));
    }
serAclose();
}
```

set

```
void set(void *address, unsigned int bit);
```

DESCRIPTION

Dynamic C may expand this call inline

Sets specified bit at memory address to 1. bit may be from 0 to 31. This is equivalent to the following expression, but more efficient:

```
*(long *)address |= 1L << bit
```

PARAMETERS

address	Address of byte containing bits 7-0
bit	Bit location where 0 represents the least significant bit

LIBRARY

UTIL.LIB

SEE ALSO

SET

SET

```
void SET(void *address, unsigned int bit);
```

DESCRIPTION

Dynamic C may expand this call inline

Sets specified bit at memory address to 1. bit may be from 0 to 31. This is equivalent to the following expression, but more efficient:

```
*(long *)address |= 1L << bit
```

PARAMETERS

address	Address of byte containing bits 7-0
bit	Bit location where 0 represents the least significant bit

LIBRARY

UTIL.LIB

SEE ALSO

set

set32kHzDivider

```
void set32kHzDivider(int setting);
```

DESCRIPTION

Sets the expanded 32kHz oscillator divider for the Rabbit 3000 processor. This function does not enable running the 32kHz oscillator instead of the main clock. This function will affect the actual rate used by the processor when the 32kHz oscillator has been enabled to run by a call to **use32kHzOsc()**.

This function is not task reentrant.

PARAMETER

setting 32kHz divider setting. The following are valid:

- **OSC32DIV_1** - don't divide 32kHz oscillator
- **OSC32DIV_2** - divide 32kHz oscillator by two
- **OSC32DIV_4** - divide 32kHz oscillator by four
- **OSC32DIV_8** - divide 32kHz oscillator by eight
- **OSC32DIV_16** - divide 32kHz oscillator by sixteen

LIBRARY

SYS.LIB

SEE ALSO

useClockDivider, useClockDivider3000, useMainOsc, use32kHzOsc

setClockModulation

```
void setClockModulation(int setting);
```

DESCRIPTION

Changes the setting of the Rabbit 3000 CPU clock moduation. Calling this function will force a 500 clock delay before the setting is changed to ensure that the previous moduation setting has cleared before the next one is set. See the *Rabbit 3000 Microprocessor User's Manual* for more details about clock modulation for EMI reduction.

PARAMETER

setting Clock modulation setting. Allowed values are:

- 0 = no modulation
- 1 = weak modulation
- 2 = strong modulation

LIBRARY

SYS.LIB

setjmp

```
int setjmp(jmp_buf env);
```

DESCRIPTION

Store the PC (program counter), SP (stack pointer) and other information about the current state into **env**. The saved information can be restored by executing **longjmp()**.

Typical usage:

```
switch (setjmp(e)) {
   case 0:           // first time
      f();           // try to execute f(), may call longjmp()
      break;         // if we get here, f() was successful
   case 1:           // to get here, f() called longjmp()
      /* do exception handling */
      break;
   case 2:           // like above, but different exception code
      ...
}
f() {
   g()
   ...
}
g() {
   ...
   longjmp(e,2);     // exception code 2, jump to setjmp()statement,
                     // setjmp() returns 2, so execute
                     // case 2 in the switch statement
}
```

PARAMETERS

env	Information about the current state

RETURN VALUE

Returns zero if it is executed. After **longjmp()** is executed, the program counter, stack pointer and etc. are restored to the state when **setjmp()** was executed the first time. However, this time **setjmp()** returns whatever value is specified by the **longjmp()** statement.

LIBRARY

SYS.LIB

SEE ALSO

longjmp

SetVectExtern2000

```
unsigned SetVectExtern2000(int priority, void *isr);
```

DESCRIPTION

Sets up the external interrupt table vectors for external interrupts 0 and 1. This function is presently used for Rabbit 2000 microprocessors because of the way they handle interrupts. Once this function is called, both external interrupts 0 and 1 must be enabled with the same priority level. The priority level should be set higher than all other interrupts currently running. (All system interrupts in the BIOS run at interrupt priority 1.) The interrupt priority is set via the control register I0CR for external interrupt 0 and I1CR for external interrupt 1.

The actual priority used by the interrupt service routine (ISR) is passed to this function.

PARAMETERS

priority	Priority the ISR should run at. Valid values are 1, 2 or 3.
isr	ISR handler address. Must be a root address.

RETURN VALUE

Address of vector table entry, or zero if **priority** is not valid.

LIBRARY

SYS.LIB

SEE ALSO

GetVectExtern2000, SetVectIntern, GetVectIntern

SetVectExtern3000

```
unsigned SetVectExtern3000(int interruptNum, void *isr);
```

DESCRIPTION

Function to set one of the external interrupt jump table entries for the Rabbit 3000 CPU. All Rabbit interrupts use jump vectors. See **SetVectIntern()** for more information.

PARAMETERS

interruptNum	External interrupt number. 0 and 1 are the only valid values.
isr	ISR handler address. Must be a root address.

RETURN VALUE

Jump address in vector table

LIBRARY

SYS.LIB

SEE ALSO

GetVectExtern3000, SetVectIntern, GetVectIntern

SetVectIntern

```
unsigned SetVectIntern(int vectNum, void *isr);
```

DESCRIPTION

Sets an internal interrupt table entry. All Rabbit interrupts use jump vectors. This function writes a **jp** instruction (0xC3) followed by the 16 bit ISR address to the appropriate location in the vector table. The location in RAM of the vector table is determined and set by the BIOS automatically at startup. The start of the table is always on a 0x100 boundary.

It is perfectly permissible to have ISRs in xmem and do long jumps to them from the vector table. It is even possible to place the entire body of the ISR in the vector table if it is 16 bytes long or less, but this function only sets up jumps to 16 bit addresses.

The following table shows the **vectNum** argument that should be used for each peripheral or RST. The offset into the vector table is also shown. The following vectors are for the Rabbit 2000 and 3000.

Peripheral or RST	vectNum	Vector Table Offset
System Management (periodic interrupt)	0x00	0x00
RST 10 instruction	0x02	0x20
RST 38 instruction	0x07	0x70
Slave Port	0x08	0x80
Timer A	0x0A	0xA0
Timer B	0x0B	0xB0
Serial Port A	0x0C	0xC0
Serial Port B	0x0D	0xD0
Serial Port C	0x0E	0xE0
Serial Port D	0x0F	0xF0

The following vectors are for the Rabbit 3000 processor only:

Peripheral or RST	vectNum	Vector Table Offset
Input Capture	0x15	0x0150
Quadrature Encoder	0x19	0x0190
Serial port E	0x1C	0x01C0
Serial port F	0x1D	0x01D0

SetVectIntern (continued)

The following three RSTs are included for completeness, but should not be set by the user normally as they are used by Dynamic C

Peripheral or RST	vectNum	Vector Table Offset
RST 18 instruction	0x03	0x30
RST 20 instruction	0x04	0x40
RST 28 instruction	0x05	0x50

PARAMETERS

vectNum Interrupt number. See the above table for valid values.

isr ISR handler address. Must be a root address.

RETURN VALUE

Address of vector table entry, or zero if **vectNum** is not valid.

LIBRARY

SYS.LIB

SEE ALSO

GetVectExtern2000, SetVectExtern2000, GetVectIntern

sin

```
float sin(float x);
```

DESCRIPTION

Computes the sine of **x**.

PARAMETERS

x Value to compute

RETURN VALUE

Sine of **x**.

LIBRARY

MATH.LIB

SEE ALSO

sinh, asin, cos, tan

sinh

```
float sinh(float x);
```

DESCRIPTION

Computes the hyperbolic sine of **x**.

PARAMETERS

x Value to compute

RETURN VALUE

The hyperbolic sine of **x**.

If **x** > 89.8 (approx.), the function returns INF and signals a range error. If **x** < –89.8 (approx.), the function returns –INF and signals a range error.

LIBRARY

MATH.LIB

SEE ALSO

sin, asin, cosh, tanh

SPIinit

```
void SPIinit ();
```

DESCRIPTION

Initialize the SPI port parameters for a serial interface only. This function does nothing for a parallel interface. A description of the values that the user may define before the **#use SPI.LIB** statement is found at the top of the library **Lib\Spi\Spi.lib**.

LIBRARY

SPI.LIB

SEE ALSO

SPIRead, SPIWrite, SPIWrRd

SPIRead

```
void SPIRead ( void *DestAddr, int ByteCount );
```

DESCRIPTION

Reads a block of bytes from the SPI port. Note: the device Chip Select must already be active. The variable **SPIxor** needs to be set to either 0x00 or 0xFF depending on whether or not the received signal needs to be inverted. Most applications will not need inversion. **SPIinit()** sets the value of **SPIxor** to 0x00.

PARAMETERS

DestAddr	Address to store the data
ByteCount	Number of bytes to read

RETURN VALUE

None.

LIBRARY

SPI.LIB

SEE ALSO

SPIinit, SPIWrite, SPIWrRd

SPIWrite

```
int SPIWrite ( void *SrcAddr, int ByteCount );
```

DESCRIPTION

Write a block of bytes to the SPI port. Note: the device Chip Select must already be active.

PARAMETERS

SrcAddr	Address of data to write
ByteCount	Number of bytes to write

RETURN VALUE

None.

LIBRARY

SPI.LIB

SEE ALSO

SPIinit, SPIRead, SPIWrRd

SPIWrRd

```
void SPIWrRd ( void *SrcAddr, void *DstAddr, int ByteCount );
```

DESCRIPTION

Read and Write a block of bytes from/to the SPI port. The device Chip Select must already be active

PARAMETERS

SrcAddr	Address of data to write.
DstAddr	Address to put received data.
ByteCount	Number of bytes to read/write. The maximum value is 255 bytes. This limit is not checked! The receive buffer MUST be at least as large as the number of bytes!

RETURN VALUE

None.

LIBRARY

SPI.LIB

SEE ALSO

SPIinit, SPIRead, SPIWrite

sprintf

```
int sprintf(char *buffer, char *format, ...);
```

DESCRIPTION

This function takes a string (pointed to by **format**), arguments of the format, and outputs the formatted string to **buffer** (pointed to by **buffer**). The user should make sure that:

- there are enough arguments after **format** to fill in the format parameters in the format string
- the types of arguments after **format** match the format fields in **format**
- the buffer is large enough to hold the longest possible formatted string

The following is a short list of possible format parameters in the format string. For more details, refer to any C language book.

%d decimal integer (expects type **int**)
%u decimal unsigned integer (expects type **unsigned int**)
%x hexadecimal integer (expects type **signed int** or **unsigned int**)
%s a string (not interpreted, expects type (**char ***))
%f a float (expects type **float**)

For example, **sprintf(buffer,"%s=%x","variable x",256);** should put the string **variable x=100** into **buffer**.

This function can be called by processes of different priorities.

PARAMETERS

buffer	Result string of the formatted string.
format	String to be formatted.
...	Format arguments.

RETURN VALUE

Number of characters written.

LIBRARY

STDIO.LIB

SEE ALSO

printf

sqrt

```
float sqrt(float x);
```

DESCRIPTION

Calculate the square root of **x**.

PARAMETERS

x Value to compute

RETURN VALUE

The square root of **x**.

LIBRARY

MATH.LIB

SEE ALSO

exp, pow, pow10

srand

```
void srand(unsigned long seed)
```

DESCRIPTION

Sets the seed value for the **rand()** function.

PARAMETER

seed This must be an odd number.

LIBRARY

MATH.LIB

SEE ALSO

rand, randb, randg

strcat

```
char *strcat(char *dst, char *src);
```

DESCRIPTION

Appends one string to another.

PARAMETERS

dst	Pointer to location to destination string.
src	Pointer to location to source string.

RETURN VALUE

Pointer to destination string.

LIBRARY

STRING.LIB

SEE ALSO

strncat

strchr

```
char *strchr(char *src, char ch);
```

DESCRIPTION

Scans a string for the first occurrence of a given character.

PARAMETERS

src	String to be scanned.
ch	Character to search

RETURN VALUE

Pointer to the first occurrence of **ch** in **src**.
NULL if **ch** is not found.

LIBRARY

STRING.LIB

SEE ALSO

strrchr, strtok

strcmp

```
int strcmp(char *str1, char *str2)
```

DESCRIPTION

Performs unsigned character by character comparison of two **NULL**-terminated strings.

PARAMETERS

str1 Pointer to string 1.

str2 Pointer to string 2.

RETURN VALUE

<0: **str1** is less than **str2** because
character in **str1** is less than corresponding character in **str2**, or
str1 is shorter than but otherwise identical to **str2**.

=0: **str1** is identical to **str2**

>0: **str1** is greater than **str2** because
character in **str1** is greater than corresponding character in **str2**, or
str2 is shorter than but otherwise identical to **str1**.

LIBRARY

STRING.LIB

SEE ALSO

strncmp, strcmpi, strncmpi

strcmpi

```
int *strcmpi(char *str1, char *str2);
```

DESCRIPTION

Performs case-insensitive unsigned character by character comparison of two **NULL**-terminated strings.

PARAMETERS

str1 Pointer to string 1.

str2 Pointer to string 2.

RETURN VALUE

<0: **str1** is less than **str2** because
character in **str1** is less than corresponding character in **str2**, or
str1 is shorter than but otherwise identical to **str2**.

=0: **str1** is identical to **str2**

>0: **str1** is greater than **str2** because
character in **str1** is greater than corresponding character in **str2**, or
str2 is shorter than but otherwise identical to **str1**.

LIBRARY

STRING.LIB

SEE ALSO

strncmpi, strncmp, strcmp

strcpy

```
char *strcpy(char *dst, char *src);
```

DESCRIPTION

Copies one string into another string including the **NULL** terminator.

PARAMETERS

dst	Pointer to location to receive string.
src	Pointer to location to supply string.

RETURN VALUE

Pointer to destination string.

LIBRARY

STRING.LIB

SEE ALSO

strncpy

strcspn

```
unsigned int strcspn(char *s1, char *s2);
```

DESCRIPTION

Scans a string for the occurrence of any of the characters in another string.

PARAMETERS

s1	String to be scanned.
s2	Character occurrence string.

RETURN VALUE

Returns the position (less one) of the first occurrence of a character in **s1** that matches any character in **s2**.

LIBRARY

STRING.LIB

SEE ALSO

strchr, strrchr, strtok

strlen

```
int strlen(char *s);
```

DESCRIPTION

Calculate the length of a string.

PARAMETERS

s	Character string

RETURN VALUE

Number of bytes in a string.

LIBRARY

STRING.LIB

strncat

```
char *strncat(char *dst, char *src, unsigned int n);
```

DESCRIPTION

Appends one string to another up to and including the **NULL** terminator or until **n** characters are transferred, followed by a **NULL** terminator.

PARAMETERS

dst	Pointer to location to receive string.
src	Pointer to location to supply string.
n	Maximum number of bytes to copy. If equal to zero, this function has no effect.

RETURN VALUE

Pointer to destination string.

LIBRARY

STRING.LIB

SEE ALSO

strcat

strncmp

```
int strncmp(char *str1, char *str2, n)
```

DESCRIPTION

Performs unsigned character by character comparison of two strings of length **n**.

PARAMETERS

str1	Pointer to string 1.
str2	Pointer to string 2.
n	Maximum number of bytes to compare. If zero, both strings are considered equal.

RETURN VALUE

<0: **str1** is less than **str2** because
char in **str1** is less than corresponding char in **str2**.

=0: **str1** is identical to **str2**

>0: **str1** is greater than **str2** because
char in **str1** is greater than corresponding char in **str2**.

LIBRARY

STRING.LIB

SEE ALSO

strcmp, strcmpi, strncmpi

strncmpi

```
int strncmpi(char *str1, char *str2, unsigned n)
```

DESCRIPTION

Performs case-insensitive unsigned character by character comparison of two strings of length **n**.

PARAMETERS

str1	Pointer to string 1.
str2	Pointer to string 2.
n	Maximum number of bytes to compare, if zero then strings are considered equal

RETURN VALUE

<0: **str1** is less than **str2** because
char in **str1** is less than corresponding char in **str2**.

=0: **str1** is identical to **str2**

>0: **str1** is greater than **str2** because
char in **str1** is greater than corresponding char in **str2**.

LIBRARY

STRING.LIB

SEE ALSO

strcmpi, strcmp, strncmp

strncpy

```
char *strncpy(char *dst, char *src, unsigned int n);
```

DESCRIPTION

Copies a given number of characters from one string to another and padding with **NULL** characters or truncating as necessary.

PARAMETERS

dst	Pointer to location to receive string.
src	Pointer to location to supply string.
n	Maximum number of bytes to copy. If equal to zero, this function has no effect.

RETURN VALUE

Pointer to destination string.

LIBRARY

STRING.LIB

SEE ALSO

strcpy

strpbrk

```
char *strpbrk(char *s1, char *s2);
```

DESCRIPTION

Scans a string for the first occurrence of any character from another string.

PARAMETERS

s1	String to be scanned.
s2	Character occurrence string.

RETURN VALUE

Pointer pointing to the first occurrence of a character contained in **s2** in **s1**. Returns **NULL** if not found.

LIBRARY

STRING.LIB

SEE ALSO

strchr, strrchr, strtok

strrchr

```
char *strrchr(char *s, int c);
```

DESCRIPTION

Similar to **strchr**, except this function searches backward from the end of **s** to the beginning.

PARAMETERS

s	String to be searched
c	Search character

RETURN VALUE

Pointer to last occurrence of **c** in **s**. If **c** is not found in **s**, return **NULL**.

LIBRARY

STRING.LIB

SEE ALSO

strchr, strcspn, strtok

strspn

```
size_t strspn(char *src, char *brk);
```

DESCRIPTION

Scans a string for the first segment in **src** containing only characters specified in **brk**.

PARAMETERS

src	String to be scanned
brk	Set of characters

RETURN VALUE

Returns the length of the segment.

LIBRARY

STRING.LIB

strstr

```
char *strstr(char *s1, char *s2);
```

DESCRIPTION

Finds a substring specified by **s2** in string **s1**.

PARAMETERS

s1	String to be scanned
s2	Substring

RETURN VALUE

Pointer pointing to the first occurrence of substring **s2** in **s1**. Returns **NULL** if **s2** is not found in **s1**.

LIBRARY

STRING.LIB

SEE ALSO

strcspn, strrchr, strtok

strtod

```
float strtod(char *s, char **tailptr);
```

DESCRIPTION

ANSI string to float conversion.

PARAMETERS

s	String to convert
tailptr	Pointer to a pointer of character. The next conversion may resume at the location specified by ***tailptr**.

RETURN VALUE

The float number.

LIBRARY

STRING.LIB

SEE ALSO

atof

strtok

```
char *strtok(char *src, char *brk);
```

DESCRIPTION

Scans **src** for tokens separated by delimiter characters specified in **brk**.

First call with non-**NULL** for **src**. Subsequent calls with **NULL** for **src** continue to search tokens in the string. If a token is found (i.e., delineators found), replace the first delimiter in **src** with a **NULL** terminator so that **src** points to a proper **NULL**-terminated token.

PARAMETERS

src	String to be scanned, must be in SRAM, cannot be a constant. In contrast, strings initialized when they are declared are stored in Flash Memory, and are treated as constants.
brk	Character delimiter

RETURN VALUE

Pointer to a token. If no delimiter (therefore no token) is found, returns **NULL**.

LIBRARY

STRING.LIB

SEE ALSO

strchr, strrchr, strstr, strcspn

strtol

```
long strtol(char *sptr, char **tailptr, int base);
```

DESCRIPTION

ANSI string to long conversion.

PARAMETERS

sptr	String to convert
tailptr	Assigned the last position of the conversion. The next conversion may resume at the location specified by ***tailptr**.
base	Indicates the radix of conversion.

RETURN VALUE

The long integer.

LIBRARY

STRING.LIB

SEE ALSO

atoi, atol

_sysIsSoftReset

```
void _sysIsSoftReset();
```

DESCRIPTION

This function determines whether this restart of the board is due to a software reset from Dynamic C or a call to **forceReset()**. If it was a soft reset, this function then does the following:

Calls **_prot_init()** to initialize the protected variable mechanisms. It is up to the user to initialize protected variables.

Calls **sysResetChain()**. The user my attach functions to this chain to perform additional startup actions (for example, initializing protected variables). If a soft reset did not take place, this function calls **_prot_recover()** to recover any protected variables.

LIBRARY

SYS.LIB

SEE ALSO

chkHardReset, chkSoftReset, chkWDTO

sysResetChain

```
void sysResetChain ( void );
```

DESCRIPTION

This is a function chain that should be used to initialize protected variables. By default, it's empty.

LIBRARY

SYS.LIB

tan

```
float tan(float x);
```

DESCRIPTION

Compute the tangent of the argument.

PARAMETERS

x Value to compute

RETURN VALUE

Returns the tangent of **x**, where $-8 \times PI \leq x \leq +8 \times PI$. If **x** is out of bounds, the function returns 0 and signals a domain error. If the value of **x** is too close to a multiple of 90° (PI/2) the function returns INF and signals a range error.

LIBRARY

MATH.LIB

SEE ALSO

atan, cos, sin, tanh

tanh

```
float tanh(float x);
```

DESCRIPTION

Computes the hyperbolic tangent of argument.

PARAMETERS

x	Value to compute

RETURN VALUE

Returns the hyperbolic tangent of **x**. If **x** > 49.9 (approx.), the function returns INF and signals a range error. If **x** < –49.9 (approx.), the function returns –INF and signals a range error.

LIBRARY

MATH.LIB

SEE ALSO

atan, cosh, sinh, tan

tm_rd

```
int tm_rd(struct tm *t);
```

DESCRIPTION

Reads the current system time from **SEC_TIMER** into the structure **t**. WARNING: The variable **SEC_TIMER** is initialized when a program is started. If you change the Real Time Clock (RTC), this variable will not be updated until you restart a program, and the **tm_rd** function will not return the time that the RTC has been reset to. The **read_rtc** function will read the actual RTC and can be used if necessary.

PARAMETERS

t Address of structure to store time data

```
struct tm {
   char tm_sec;     // seconds 0-59
   char tm_min;     // 0-59
   char tm_hour;    // 0-23
   char tm_mday;    // 1-31
   char tm_mon;     // 1-12
   char tm_year;    // 80-147 (1980-2047)
   char tm_wday;    // 0-6 0==Sunday
};
```

RETURN VALUE

0: Successful.
-1: Clock read failed.

LIBRARY

RTCLOCK.LIB

SEE ALSO

mktm, mktime, tm_wr

tm_wr

```
int tm_wr(struct tm *t);
```

DESCRIPTION

Sets the system time from a **tm** struct. It is important to note that although **tm_rd()** reads the **SEC_TIMER** variable, not the RTC, **tm_wr()** writes to the RTC directly, and **SEC_TIMER** is not changed until the program is restarted. The reason for this is so that the **DelaySec()** function continues to work correctly after setting the system time. To make **tm_rd()** match the new time written to the RTC without restarting the program, the following should be done:

```
tm_wr(tm);
SEC_TIMER = mktime(tm);
```

But this could cause problems if a **waitfor(DelaySec(n))** is pending completion in a cooperative multitasking program or if the **SEC_TIMER** variable is being used in another way the user, so user beware.

PARAMETERS

t Pointer to structure to read date and time from.

```
struct tm {
   char tm_sec;     // seconds 0-59
   char tm_min;     // 0-59
   char tm_hour;    // 0-23
   char tm_mday;    // 1-31
   char tm_mon;     // 1-12
   char tm_year;    // 80-147 (1980-2047)
   char tm_wday;    // 0-6 0==Sunday
};
```

RETURN VALUE

0: Success .
-1: Failure.

LIBRARY

RTCLOCK.LIB

SEE ALSO

mktm, mktime, tm_rd

tolower

```
int tolower(int c);
```

DESCRIPTION

Convert alphabetic character to lower case.

PARAMETERS

c Character to convert

RETURN VALUE

Lower case alphabetic character.

LIBRARY

STRING.LIB

SEE ALSO

toupper, isupper, islower

toupper

```
int toupper(int c);
```

DESCRIPTION

Convert alphabetic character to uppercase.

PARAMETERS

c Character to convert

RETURN VALUE

Upper case alphabetic character.

LIBRARY

STRING.LIB

SEE ALSO

tolower, isupper, islower

updateTimers

```
void updateTimers();
```

DESCRIPTION

Updates the values of **TICK_TIMER**, **MS_TIMER**, and **SEC_TIMER** while running off the 32 kHz oscillator. Since the periodic interrupt is disabled when running at 32 kHz, these values will not updated unless this function is called.

LIBRARY

SYS.LIB

SEE ALSO

useMainOsc, use32kHzOsc

use32kHzOsc

```
void use32kHzOsc();
```

DESCRIPTION

Sets the Rabbit processor to use the 32kHz real-time clock oscillator for both the CPU and peripheral clock, and shuts off the main oscillator. If this is already set, there is no effect. This mode should provide greatly reduced power consumption. Serial communications will be lost since typical baud rates cannot be made from a 32kHz clock. Also note that this function disables the periodic interrupt, so **waitfor** and related statements will not work properly (although costatements in general will still work). In addition, the values in **TICK_TIMER**, **MS_TIMER**, and **SEC_TIMER** will not be updated unless you call the function **updateTimers()** frequently in your code. In addition, you will need to call **hitwd()** periodically to hit the hardware watchdog timer since the periodic interrupt normally handles that, or disable the watchdog timer before calling this function. The watchdog can be disabled with **Disable_HW_WDT()**.

use32kHzOsc() is not task reentrant.

LIBRARY

SYS.LIB

SEE ALSO

useMainOsc, useClockDivider, updateTimers

useClockDivider

```
void useClockDivider();
```

DESCRIPTION

Sets the Rabbit processor to use the main oscillator divided by 8 for the CPU (but not the peripheral clock). If this is already set, there is no effect. Because the peripheral clock is not affected, serial communications should still work. This function also enables the periodic interrupt in case it was disabled by a call to **use32kHzOsc()**. This function is not task reentrant.

LIBRARY

SYS.LIB

SEE ALSO

useMainOsc, use32kHzOsc

useClockDivider3000

```
void useClockDivider3000(int setting);
```

DESCRIPTION

Sets the expanded clock divider options for the Rabbit 3000 processor. This function will also affect the peripheral clock—use **useClockDivider()** to divide the processor clock by eight but not affect the peripheral clock. Target communications will be lost after changing this setting because of the baud rate change. This function also enables the periodic interrupt in case it was disabled by a call to **user32kHzOsc()**.

This function is not task reentrant.

PARAMETER

setting Divider setting. The following are valid:

- **CLKDIV_2** - divide main processor clock by two
- **CLKDIV_4** - divide main processor clock by four
- **CLKDIV_6** - divide main processor clock by six
- **CLKDIV_8** - divide main processor clock by eight

RETURN VALUE

None.

LIBRARY

SYS.LIB

SEE ALSO

useClockDivider, useMainOsc, use32kHzOsc, set32kHzDivider

useMainOsc

```
void useMainOsc();
```

DESCRIPTION

Sets the Rabbit processor to use the main oscillator for both the CPU and peripheral clock. If this is already set, there is no effect. This function also enables the periodic interrupt in case it was disabled by a call to **use32kHzOsc()**, and updates the **TICK_TIMER**, **MS_TIMER**, and **SEC_TIMER** variables from the real-time clock. This function is not task reentrant.

LIBRARY

sys.lib

SEE ALSO

use32kHzOsc, useClockDivider

utoa

```
char *utoa(unsigned value, char *buf);
```

DESCRIPTION

Places up to 5 digit character string at ***buf** representing value of unsigned number. Suppresses leading zeros, but leaves one zero digit for value = 0. Max = 65535. 73 program bytes.

PARAMETERS

value 16-bit number to convert

buf Character string of converted number

RETURN VALUE

Pointer to **NULL** at end of string.

LIBRARY

STDIO.LIB

SEE ALSO

itoa, htoa, ltoa

VdGetFreeWd

```
int VdGetFreeWd(char count);
```

DESCRIPTION

Returns a free virtual watchdog and initializes that watchdog so that the virtual driver begins counting it down from **count**. The number of virtual watchdogs available is determined by **N_WATCHDOG**, which is 5 by default, but can be defined by the user: **#define N_WATCHDOG 10**. The virtual driver is called every 0.00048828125 sec. On every 128th call to it (62.5 ms), the virtual watchdogs are counted down. If any virtual watchdog reaches 0, this is a fatal error. Once a virtual watchdog is active, it should reset periodically with a call to **VdHitWd** to prevent this. The count is decremented, tested and, if 0, a fatal error occurs.

PARAMETERS

count 1 < **count** <= 255

RETURN VALUE

Integer id number of an unused virtual watchdog timer.

LIBRARY

VDRIVER.LIB

VdHitWd

```
int VdHitWd(int ndog);
```

DESCRIPTION

Resets virtual watchdog counter to N counts where N is the argument to the call to **VdGetFreeWd()** that obtained the virtual watchdog **ndog**. The virtual driver counts down watchdogs every 62.5 ms. If a virtual watchdog reaches 0, this is a fatal error. Once a virtual watchdog is active it should reset periodically with a call to **VdHitWd()** to prevent this. If count = 2 the **VdHitWd()** will need to be called again for virtual watchdog **ndog** within 62.5 ms. If count = 255, **VdHitWd()** will need to be called again for virtual watchdog **ndog** within 15.9375 seconds.

PARAMETERS

ndog Id of virtual watchdog returned by **VdGetFreeWd()**

LIBRARY

VDRIVER.LIB

VdInit

```
void VdInit(void);
```

DESCRIPTION

Initializes the Virtual Driver for all Rabbit boards. Supports **DelayMs()**, **DelaySec()**, **DelayTick()**. **VdInit()** is called by the BIOS unless it has been disabled.

LIBRARY

VDRIVER.LIB

VdReleaseWd

```
int VdReleaseWd(int ndog);
```

DESCRIPTION

Deactivates a virtual watchdog and makes it available for **VdGetFreeWd()**.

PARAMETERS

ndog Handle returned by **VdGetFreeWd()**

RETURN VALUE

0: **ndog** out of range.
1: Success.

LIBRARY

VDRIVER.LIB

EXAMPLE

```
// VdReleaseWd virtual watchdog example
main() {
   int wd;                 // handle for a virtual watchdog
   unsigned long tm;
   tm = SEC_TIMER;
   wd = VdGetFreeWd(255);   // wd activated, 9 virtual watchdogs
                            // now available. wd must be hit
                            // at least every 15.875 seconds
   while(SEC_TIMER - tm < 60) {    // let it run for a minute
      VdHitWd(wd);          // decrements counter corresponding to
                            // wd reset to 12
   }
   VdReleaseWd(wd)          // now there are 10 virtual
                            // watchdogs available
}
```

WriteFlash2

```
int WriteFlash2(unsigned long flashDst, void* rootSrc, int
   len);
```

DESCRIPTION

Write **len** bytes from **rootSrc** to physical address **flashDst** on the 2nd flash device. The source must be in root. The **flashDst** address must be in the range 0x00040000-0x0007FFFF. (256kb is the maximum size visible on the second flash by this function).

This function is not reentrant.

> **Note:** This function should NOT be used if you are using the second flash device for a flash file system. This function is extremely dangerous when used with large-sector flash. Don't do it.

PARAMETERS

flashDst	Physical address of the flash destination
rootSrc	Pointer to the root source
len	Number of bytes to write

RETURN VALUE

0: Success.
-1: Attempt to write non-2nd flash area, nothing written.
-2: **rootsrc** not in root.
-3: Time out while writing flash.
-4: Attempt to write to ID block
-5: Sector erase needed; write aborted

LIBRARY

XMEM.LIB

write_rtc

```
void write_rtc(unsigned long int time);
```

DESCRIPTION

Writes a 32 bit seconds value to the RTC, zeros other bits. This function does not stop or delay periodic interrupt. It does not affect the **SEC_TIMER** or **MS_TIMER** variables.

PARAMETERS

time	32-bit value representing the number of seconds since January 1, 1980.

LIBRARY

RTCLOCK.C

SEE ALSO

read_rtc

writeUserBlock

```
int writeUserBlock(int addr, void *source, int numbytes);
```

DESCRIPTION

Z-World boards have a System ID Block located on the primary flash. (See the *Rabbit 2000 Microprocessor Designer's Handbook* for more information on the System ID Block.) Version 2 and later of this ID block has a pointer to a User ID Block: a place intended for storing calibration constants, passwords, and other non-volatile data.

The User ID Block is recommended for storing all non-file data. This is where calibration constants are stored for boards with analog I/O. Space here is limited to as small as **(8K - sizeof(SysIDBlock))** bytes, or less, if there are calibration constants.

writeUserBlock() writes a number of bytes from root memory to the user block. This block is protected from normal writes to the flash device and can only be accessed through this function.

> **Note:** See the manual for your particular board for more information before overwriting any part of the user block.

Backwards Compatibility:

If the version of the System ID block doesn't support the User ID block, or no System ID block is present, then 8K bytes starting 16K bytes from the top of the primary flash are designated the User ID block area. However, to prevent errors arisng from incompatible large sector configuartions, this will only work if the flash type is small sector. Z-World manufactured boards with large sector flash will have valid System and User ID blocks, so this should not be problem on Z-World boards.

If users create boards with large sector flash, they must install System ID blocks version 2 or greater to use or modify this function.

PARAMETERS

addr	Address offset in user block to write to.
source	Pointer to source to copy data from.
numbytes	Number of bytes to copy.

RETURN VALUE

0: Successful.
-1: Invalid address or range.

LIBRARY

IDBLOCK.LIB

SEE ALSO

readUserBlock

WrPortE

```
void WrPortE(unsigned int port, char *portshadow, int
   data_value);
```

DESCRIPTION

Writes an external I/O register with 8 bits and updates shadow for that register. The variable names must be of the form **port** and **portshadow** for the most efficient operation. A **NULL** pointer may be substituted if shadow support is not desired or needed.

PARAMETERS

port	Address of external data register.
portshadow	Reference pointer to a variable shadowing the register data. Substitute with **NULL** pointer (or 0) if shadowing is not required.
data_value	Value to be written to the data register

LIBRARY

SYSIO.LIB

SEE ALSO

RdPortI, BitRdPortI, WrPortI, BitWrPortI, RdPortE, BitRdPortE, BitWrPortE

WrPortI

```
void WrPortI(int port, char *portshadow, int data_value);
```

DESCRIPTION

Writes an internal I/O register with 8 bits and updates shadow for that register.

PARAMETERS

port	Address of data register.
portshadow	Reference pointer to a variable shadowing the register data. Substitute with **NULL** pointer (or 0) if shadowing is not required.
data_value	Value to be written to the data register

LIBRARY

SYSIO.LIB

SEE ALSO

RdPortI, BitRdPortI, BitRdPortE, BitWrPortI, RdPortE, WrPortE, BitWrPortE

xalloc

```
long xalloc(long sz)
```

DESCRIPTION

Allocates the specified number of bytes in extended memory.

PARAMETERS

sz	Number of bytes to allocate.

RETURN VALUE

The 20-bit physical address of the allocated data: Success.
0: Failure.

Note: This return value cannot be used with pointer arithmetic.

LIBRARY

STACK.LIB

SEE ALSO

root2xmem, xmem2root

xmem2root

```
int xmem2root(void *dest, unsigned long int src, unsigned int
   len);
```

DESCRIPTION

Stores **len** characters from physical address **src** to logical address **dest**.

PARAMETERS

dest	Logical address
src	Physical address
len	Numbers of bytes

RETURN VALUE

0: Success.
-1: Attempt to write flash memory area, nothing written.
-2: Destination not all in root.

LIBRARY

XMEM.LIB

SEE ALSO

root2xmem, xalloc

xmem2xmem

```
int xmem2xmem(unsigned long dest, unsigned long src, unsigned
   len);
```

DESCRIPTION

Stores **len** characters from physical address **src** to physical address **dest**.

PARAMETERS

dest	Physical address of destination
src	Physical address of source data
len	Length of source data in bytes

RETURN VALUE

0: Success.
-1: Attempt to write Flash Memory area, nothing written.

LIBRARY

XMEM.LIB

Dynamic C Function Reference Manual

Part Number 019-0113 • 020409–A • Printed in U.S.A.

Notice to Users

Z-WORLD PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE-SUPPORT DEVICES OR SYSTEMS UNLESS A SPECIFIC WRITTEN AGREEMENT REGARDING SUCH INTENDED USE IS ENTERED INTO BETWEEN THE CUSTOMER AND Z-WORLD PRIOR TO USE. Life-support devices or systems are devices or systems intended for surgical implantation into the body or to sustain life, and whose failure to perform, when properly used in accordance with instructions for use provided in the labeling and user's manual, can be reasonably expected to result in significant injury.

No complex software or hardware system is perfect. Bugs are always present in a system of any size. In order to prevent danger to life or property, it is the responsibility of the system designer to incorporate redundant protective mechanisms appropriate to the risk involved.

Trademarks

Dynamic C® is a registered trademark of Z-World Inc.

Windows® is a registered trademark of Microsoft Corporation

Z-World, Inc.

2900 Spafford Street
Davis, California 95616-6800
USA

Telephone: (530) 757-3737
Fax: (530) 757-3792
www.zworld.com

Z-WORLD SOFTWARE END USER LICENSE AGREEMENT

IMPORTANT-READ CAREFULLY: BY INSTALLING, COPYING OR OTHERWISE USING THE ENCLOSED Z-WORLD,INC. ("Z-WORLD") DYNAMIC C SOFTWARE, WHICH INCLUDES COMPUTER SOFTWARE ("SOFTWARE") AND MAY INCLUDE ASSOCIATED MEDIA, PRINTED MATERIALS, AND "ONLINE" OR ELECTRONIC DOCUMENTATION ("DOCUMENTATION"), YOU (ON BEHALF OF YOURSELF OR AS AN AUTHORIZED REPRESENTATIVE ON BEHALF OF AN ENTITY) AGREE TO ALL THE TERMS OF THIS END USER LICENSE AGREEMENT ("LICENSE") REGARDING YOUR USE OF THE SOFTWARE. IF YOU DO NOT AGREE WITH ALL OF THE TERMS OF THIS LICENSE, DO NOT INSTALL, COPY OR OTHERWISE USE THE SOFTWARE AND IMMEDIATELY CONTACT Z-WORLD FOR RETURN OF THE SOFTWARE AND A REFUND OF THE PURCHASE PRICE FOR THE SOFTWARE.

We are sorry about the formality of the language below, which our lawyers tell us we need to include to protect our legal rights. If You have any questions, write or call Z-World at (530) 757-4616, 2900 Spafford Street, Davis, California 95616.

1. **Definitions**. In addition to the definitions stated in the first paragraph of this document, capitalized words used in this License shall have the following meanings:

   1.1 "Qualified Applications" means an application program developed using the Software and that links with the development libraries of the Software.

   1.1.1 "Qualified Applications" is amended to include application programs developed using the Softools WinIDE program for Rabbit processors available from Softools, Inc.

   1.1.2 The MicroC/OS-II (uC/OS-II) library and sample code and the Point-to-Point Protocol (PPP) library are not included in this amendment.

   1.1.3 Excluding the exceptions in 1.1.2, library and sample code provided with the Software may be modified for use with the Softools WinIDE program in Qualified Systems as defined in 1.2. All other Restrictions specified by this license agreement remain in force.

   1.2 "Qualified Systems" means a microprocessor-based computer system which is either (i) manufactured by, for or under license from Z-WORLD, or (ii) based on the Rabbit 2000 microprocessor or the Rabbit 3000 microprocessor. Qualified Systems may not be (a) designed or intended to be re-programmable by your customer using the Software, or (b) competitive with Z-WORLD products, except as otherwise stated in a written agreement between Z-World and the system manufacturer. Such written agreement may require an end user to pay run time royalties to Z-World.

2. **License**. Z-WORLD grants to You a nonexclusive, nontransferable license to (i) use and reproduce the Software, solely for internal purposes and only for the number of users for which You have purchased licenses for (the "Users") and not for redistribution or resale; (ii) use and reproduce the Software solely to develop the Qualified Applications; and (iii) use, reproduce and distribute, the Qualified Applications, in object code only, to end users solely for use on Qualified Systems; provided, however, any agreement entered into between You and such end users with respect to a Qualified Application is no less protective of Z-Worlds intellectual property rights than the terms and conditions of this License. (iv) use and distribute with Qualified Applications and Qualified Systems the program files distributed with Dynamic C named **RFU.EXE**, **PILOT.BIN**, and **COLDLOAD.BIN** in their unaltered forms.
3. **Restrictions**. Except as otherwise stated, You may not, nor permit anyone else to, decompile, reverse engineer, disassemble or otherwise attempt to reconstruct or discover the source code of the Software, alter, merge, modify, translate, adapt in any way, prepare any derivative work based upon the Software, rent, lease network, loan, distribute or otherwise transfer the Software or any copy thereof. You shall not make copies of the copyrighted Software and/or documentation without the prior written permission of Z-WORLD; provided that, You may make one (1) hard copy of such documentation for each User and a reasonable number of back-up copies for Your own archival purposes. You may not use copies of the Software as part of a benchmark or comparison test against other similar products in order to produce results strictly for purposes of comparison. The Software contains copyrighted material, trade secrets and other proprietary material of Z-WORLD and/or its licensors and You must reproduce, on each copy of the Software, all copyright notices and any other proprietary legends that appear on or in the original copy of the Software. Except for the limited license granted above, Z-WORLD retains all right, title and interest in and to all intellectual property rights embodied in the Software, including but not limited to, patents, copyrights and trade secrets.
4. **Export Law Assurances**. You agree and certify that neither the Software nor any other technical data received from Z-WORLD, nor the direct product thereof, will be exported outside the United States or re-exported except as authorized and as permitted by the laws and regulations of the United States and/or the laws and regulations of the jurisdiction, (if other than the United States) in which You rightfully obtained the Software. The Software may not be exported to any of the following countries: Cuba, Iran, Iraq, Libya, North Korea, Sudan, or Syria.
5. **Government End Users**. If You are acquiring the Software on behalf of any unit or agency of the United States Government, the following provisions apply. The Government agrees: (i)if the Software is supplied to the Department of Defense ("DOD"), the Software is classified as "Commercial Computer Software" and the Government is acquiring only "restricted rights" in the Software and its documentation as that term is defined in Clause 252.227-7013(c)(1) of the DFARS; and (ii) if the Software is supplied to any unit or agency of the United States Government other than DOD, the Government's rights in the Software and its documentation will be as defined in Clause 52.227-19(c)(2) of the FAR or, in the case of NASA, in Clause 18-52.227-86(d) of the NASA Supplement to the FAR.

6. **Disclaimer of Warranty**. You expressly acknowledge and agree that the use of the Software and its documentation is at Your sole risk. THE SOFTWARE, DOCUMENTATION, AND TECHNICAL SUPPORT ARE PROVIDED ON AN "AS IS" BASIS AND WITHOUT WARRANTY OF ANY KIND. Information regarding any third party services included in this package is provided as a convenience only, without any warranty by Z-WORLD, and will be governed solely by the terms agreed upon between You and the third party providing such services. Z-WORLD AND ITS LICENSORS EXPRESSLY DISCLAIM ALL WARRANTIES, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT OF THIRD PARTY RIGHTS. Z-WORLD DOES NOT WARRANT THAT THE FUNCTIONS CONTAINED IN THE SOFTWARE WILL MEET YOUR REQUIREMENTS, OR THAT THE OPERATION OF THE SOFTWARE WILL BE UNINTERRUPTED OR ERROR-FREE, OR THAT DEFECTS IN THE SOFTWARE WILL BE CORRECTED. FURTHERMORE, Z-WORLD DOES NOT WARRANT OR MAKE ANY REPRESENTATIONS REGARDING THE USE OR THE RESULTS OF THE SOFTWARE IN TERMS OF ITS CORRECTNESS, ACCURACY, RELIABILITY OR OTHERWISE. NO ORAL OR WRITTEN INFORMATION OR ADVICE GIVEN BY Z-WORLD OR ITS AUTHORIZED REPRESENTATIVES SHALL CREATE A WARRANTY OR IN ANY WAY INCREASE THE SCOPE OF THIS WARRANTY. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO THE ABOVE EXCLUSION MAY NOT APPLY TO YOU.
7. **Limitation of Liability**. YOU AGREE THAT UNDER NO CIRCUMSTANCES, INCLUDING NEGLIGENCE, SHALL Z-WORLD BE LIABLE FOR ANY INCIDENTAL, SPECIAL OR CONSEQUENTIAL DAMAGES (INCLUDING DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION AND THE LIKE) ARISING OUT OF THE USE AND/OR INABILITY TO USE THE SOFTWARE, EVEN IF Z-WORLD OR ITS AUTHORIZED REPRESENTATIVE HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. SOME JURISDICTIONS DO NOT ALLOW THE LIMITATION OR EXCLUSION OF LIABILITY FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES SO THE ABOVE LIMITATION OR EXCLUSION MAY NOT APPLY TO YOU. IN NO EVENT SHALL Z-WORLDS TOTAL LIABILITY TO YOU FOR ALL DAMAGES, LOSSES, AND CAUSES OF ACTION (WHETHER IN CONTRACT, TORT, INCLUDING NEGLIGENCE, OR OTHERWISE) EXCEED THE AMOUNT PAID BY YOU FOR THE SOFTWARE.
8. **Termination**. This License is effective for the duration of the copyright in the Software unless terminated. You may terminate this License at any time by destroying all copies of the Software and its documentation. This License will terminate immediately without notice from Z-WORLD if You fail to comply with any provision of this License. Upon termination, You must destroy all copies of the Software and its documentation. Except for Section 2 ("License"), all Sections of this Agreement shall survive any expiration or termination of this License.

9. **General Provisions**. No delay or failure to take action under this License will constitute a waiver unless expressly waived in writing, signed by a duly authorized representative of Z-WORLD, and no single waiver will constitute a continuing or subsequent waiver. This License may not be assigned, sublicensed or otherwise transferred by You, by operation of law or otherwise, without Z-WORLD's prior written consent. This License shall be governed by and construed in accordance with the laws of the United States and the State of California, exclusive of the conflicts of laws principles. The United Nations Convention on Contracts for the International Sale of Goods shall not apply to this License. If for any reason a court of competent jurisdiction finds any provision of this License, or portion thereof, to be unenforceable, that provision of the License shall be enforced to the maximum extent permissible so as to affect the intent of the parties, and the remainder of this License shall continue in full force and effect. This License constitutes the entire agreement between the parties with respect to the use of the Software and its documentation, and supersedes all prior or contemporaneous understandings or agreements, written or oral, regarding such subject matter. There shall be no contract for purchase or sale of the Software except upon the terms and conditions specified herein. Any additional or different terms or conditions proposed by You or contained in any purchase order are hereby rejected and shall be of no force and effect unless expressly agreed to in writing by Z-WORLD. No amendment to or modification of this License will be binding unless in writing and signed by a duly authorized representative of Z-WORLD.