CrossWorks Tools Library

Version: 3.1

Contents

CrossWorks Tools Library

CrossWorks Tools Library

About the CrossWorks Tools Library

The *CrossWorks Tools Library* presents a standardized API for delivering high-quality example code for a wide range of microcontrollers and evaluation boards. Additional components that integrate with the Tools Library are:

- *CrossWorks Tools Library*: provides add-ons for CTL such as read-write locks and ring buffers.
- *CrossWorks Device Library*: provides drivers for common digital sensors, such as accelerometers, gyroscopes, magnetometers, and so on.
- *CrossWorks Graphics Library*: is a library of simple graphics functions for readily-available LCD controllers.
- *CrossWorks TCP/IP Library*: provides TCP/IP networking for integrated and external network controllers on memory-constrained microcontrollers.
- *CrossWorks Mass Storage Library*: provides a FAT-based file system for mass storage on SD and MMC cards, or any device with a block-based interface.
- *CrossWorks Shield Library*: provides drivers for a range of Arduino-style shields.
- *CrossWorks CoreBASIC Library*: provides a full-featured, network-enabled BASIC interpreter which demonstrates the capabilities of these libraries.

Architecture

The *CrossWorks Tools Library* is one part of the *CrossWorks Target Library*. Many of the low-level functions provided by the target library are built using features of the *CrossWorks Tasking Library* for multi-threaded operation.

Delivery format

The *CrossWorks Tools Library* is delivered in source form.

Feedback

This facility is a work in progress and may undergo rapid change. If you have comments, observations, suggestions, or problems, please feel free to air them on the **CrossWorks Target and Platform API** discussion forum.

License

The following terms apply to the Rowley Associates Tools Library.

CrossWorks Tools Library

Introduction

About the CrossWorks Tools Library

<ctl_tools.h>

Overview

A toolkit of additional CTL functions not included in the core CTL API.

Definitions in this file are here temporarily, or perhaps permanently. They may move into the main CrossWorks V3 and CTL V3 release, or they might just be parked here forever.

API Summary

Errors	
CTL_CORE_ERROR_t	Common errors
CTL_ERROR	Construct error code
CTL_ERROR_INDICATOR	Error indicator bit pattern
CTL_STATUS_t	API completion status
ctl_normalize_status	Normalize status
ctl_register_core_error_decoder	Register core error decoder
ctl_strerror	Decodes an error code
Data	
CTL_BYTE_SWAP_16b_LIT	Byte-swap 16-bit word
CTL_BYTE_SWAP_32b_LIT	Byte-swap 16-bit word
ctl_atomic_decrement	Decrement counter
ctl_atomic_increment	Increment counter
ctl_byte_swap_16b	Byte-swap 16-bit word
ctl_byte_swap_32b	Byte-swap 32-bit word
ctl_load_16b_be	Load a 16-bit big-endian value
ctl_load_16b_le	Load a 16-bit little-endian value
ctl_load_24b_be	Load a 24-bit big-endian value
ctl_load_24b_le	Load a 24-bit little-endian value
ctl_load_32b_be	Load a 32-bit big-endian value
ctl_load_32b_le	Load a 32-bit little-endian value
Streams	
CTL_STREAM_CB_t	Stream control block
CTL_STREAM_DRIVER_t	Stream driver
CTL_STREAM_NULL	Null stream
CTL_STREAM_STANDARD_DEBUG	Standard debug stream

CrossWorks Tools Library

CTL_STREAM_STANDARD_ERROR	Standard error stream
CTL_STREAM_STANDARD_INPUT	Standard input stream
CTL_STREAM_STANDARD_OUTPUT	Standard output stream
CTL_STREAM_t	A generic stream
ctl_stream_alloc	Allocate a stream control block
ctl_stream_assign_standard	Assign standard stream
ctl_stream_close	Close stream
ctl_stream_find_cb	Find stream control block
ctl_stream_flush	Flush stream
ctl_stream_free_cb	Free stream control block
ctl_stream_is_null	Is stream null?
ctl_stream_is_open	Is stream open?
ctl_stream_signature	Get stream signature
ctl_stream_unmap	Unmap a stream
Formatted I/O	
ctl_printf	Print to standard output stream
ctl_puts	Put string to standard output stream
ctl_stream_getchar	Read single character from stream
ctl_stream_printf	Print to stream
ctl_stream_read	Read data from stream
ctl_stream_read_line	Read line from stream
ctl_stream_vprintf	Print to stream using variable argument context
ctl_stream_write	Write data to stream
ctl_vprintf	Print to standard output stream using variable argument context
Read-write locks	
CTL_RWLOCK_t	Read-write lock type
ctl_rwlock_claim_read_lock	Claim read lock
ctl_rwlock_claim_write_lock	Claim write lock
ctl_rwlock_init	Initialize read-write lock
ctl_rwlock_read_lock	Attempt to claim read lock, with timeout
ctl_rwlock_read_lock_uc	Attempt to claim read lock, unconditional
ctl_rwlock_release_read_lock	Release read lock
ctl_rwlock_release_write_lock	Release write lock
ctl_rwlock_write_lock	Attempt to claim write lock

Time	
ctl_delay	Delay a number of CTL ticks
platform_spin_delay_ms	Delay a number of milliseconds
platform_spin_delay_us	Delay a number of microseconds
Memory	
CTL_MEMORY_ALLOCATOR_t	Memory allocator
ctl_system_memory_allocator	System memory allocator
Utility	
ctl_print_status	Prints a CTL status code
ctl_task_decode_state	Decode task running state
Statistics	
CTL_STATS_DESCRIPTOR_t	A statistics descriptor
CTL_STATS_GROUP_t	A statistics group
ctl_stats_group	Get statistics group
ctl_stats_group_count	Number of registered statistics groups
ctl_stats_print	Print statistics
ctl_stats_register	Register statistics group

CTL_BYTE_SWAP_16b_LIT

Synopsis

```
#define CTL_BYTE_SWAP_16b_LIT(N) \
   ((((N) & 0xFF) << 8) | \
   (((N) & 0xFF00) >> 8))
```

Description

CTL_BYTE_SWAP_16b_LIT reverses the byte order of the two least significant 16 bits of **N** and returns that value. You can use this macro to initialize a static variable with a byte-swapped constant.

See Also

CTL_BYTE_SWAP_32b_LIT, ctl_byte_swap_16b

CTL_BYTE_SWAP_32b_LIT

Synopsis

```
#define CTL_BYTE_SWAP_32b_LIT(N) \
   ((((N) & 0xFF) << 24) | \
   (((N) & 0xFF00) << 8) | \
   (((N) & 0xFF0000) >> 8) | \
   (((N) & 0xFF000000) >> 24))
```

Description

CTL_BYTE_SWAP_32b_LIT reverses the byte order of the four bytes of **N** and returns that value. You can use this macro to initialize a static variable with a byte-swapped constant.

See Also

CTL_BYTE_SWAP_16b_LIT, ctl_byte_swap_32b

CTL_CORE_ERROR_t

Synopsis

```
typedef enum {
  CTL_NO_ERROR,
  CTL_OK,
  CTL_EOF,
  CTL_GENERAL_ERROR,
  CTL_OUT_OF_MEMORY,
  CTL_PARAMETER_ERROR,
  CTL_TIMEOUT_ERROR,
  CTL_UNSUPPORTED_OPERATION,
  CTL_INTERNAL_ERROR,
  CTL_READ_ERROR,
  CTL_WRITE_ERROR,
  CTL_DEVICE_NOT_READY,
  CTL_ADDRESS_IN_USE,
  CTL_HARDWARE_ERROR,
  CTL_PROTOCOL_ERROR,
  CTL_DEVICE_NOT_SELECTED,
  CTL_DEVICE_NOT_RESPONDING,
  CTL_INCORRECT_DEVICE,
  CTL_CONFIGURATION_ERROR,
  CTL_DEVICE_IN_USE,
  CTL_ARBITRATION_LOST,
  CTL_BAD_STREAM
} CTL_CORE_ERROR_t;
```

Description

CTL_CORE_ERROR_t defines a set of common, non-specific errors that client code, including libraries supplied in CrossWorks, might like to for a **CTL_STATUS_t**.

Of particular note is **CTL_NO_ERROR**, defined as zero, that indicates there is no error to report. This is the same as **CTL_OK**.

See Also

CTL_STATUS_t

CTL_ERROR

Synopsis

```
#define CTL_ERROR(CODE)  (CTL_ERROR_INDICATOR | (CODE))
```

Description

CTL_ERROR constructs an error code with indicator **CODE**.

CTL_ERROR_INDICATOR

Synopsis

```
#define CTL_ERROR_INDICATOR ((int)(0xFFFFF000))
```

Description

CTL_ERROR_INDICATOR is bitwise-or'd with a 12-bit error code to indicate an error. It evaluates to a negative integer.

See Also

CTL_ERROR

CTL_MEMORY_ALLOCATOR_t

Synopsis

```
typedef struct {
  void *(*alloc)(size_t);
  void (*free)(void *);
} CTL_MEMORY_ALLOCATOR_t;
```

Description

CTL_MEMORY_ALLOCATOR_t defines a standard memory allocator that can allocate and free memory.

Structure

`alloc`

Allocates memory.

`free`

Frees memory.

CrossWorks Tools Library

CTL_RWLOCK_t

Synopsis

```
typedef struct {
  CTL_SEMAPHORE_t readers;
  CTL_MUTEX_t writer;
  unsigned max_readers;
} CTL_RWLOCK_t;
```

Description

CTL_RWLOCK_t defines a read-write lock. All members are private.

CTL_STATS_DESCRIPTOR_t

Synopsis

```
typedef struct {
  const char *const name;
  volatile long *data;
} CTL_STATS_DESCRIPTOR_t;
```

Description

CTL_STATS_DESCRIPTOR_t describes a single statistic that can be monitored.

Structure

name

A pointer to the string that contains the user-facing name of the statistic.

data

A pointer to the statistic data item.

CTL_STATS_GROUP_t

Synopsis

```
typedef struct {
  const char *group;
  const CTL_STATS_DESCRIPTOR_t *desc;
} CTL_STATS_GROUP_t;
```

Description

CTL_STATS_GROUP_t describes a group of statistics.

Structure

group

A pointer to the string that contains the user-facing name of the statistic group.

desc

A pointer to the first element of the array of statistics.

CTL_STATUS_t

Synopsis

```
typedef long CTL_STATUS_t;
```

Description

CTL_STATUS_t is the new status code mechanism for the add-on libraries, such as the mass storage library, TCP/IP networking library, USB device library, sensor library, and the set of drivers associated with them.

All error codes indicating errors are negative. A zero indicates a "no error" condition. API calls may return zero to indicate success if they only need to indicate success or failure. API calls may wish to return more information, which is always positive.

CTL_STREAM_CB_t

Synopsis

```
typedef struct {
  CTL_STREAM_t handle;
  unsigned long signature;
  const CTL_STREAM_DRIVER_t *d;
  CTL_STREAM_CB_t *__next;
} CTL_STREAM_CB_t;
```

Description

CTL_STREAM_CB_t defines the base control block for a CTL stream.

Structure

handle

The stream handle allocated by `ctl_stream_alloc`.

signature

A unique pattern that differentiates types of stream. For instance, a TCP socket stream has a different signature from a file stream, and both of which are different from a UART stream.

d

The methods to read and write the stream.

__next

A private member maintained by CTL that points to the next stream in the list of all streams.

See Also

ctl_stream_alloc.

CTL_STREAM_DRIVER_t

Synopsis

```
typedef struct {
  CTL_STATUS_t (*__read)(CTL_STREAM_t , void *, size_t);
  CTL_STATUS_t (*__write)(CTL_STREAM_t , const void *, size_t);
  CTL_STATUS_t (*__flush)(CTL_STREAM_t);
  CTL_STATUS_t (*__close)(CTL_STREAM_t);
} CTL_STREAM_DRIVER_t;
```

Description

CTL_STREAM_DRIVER_t defines the methods that implement I/O to a stream.

Structure

__read

Read from stream.

__write

Write to stream.

__flush

Flush any buffered data associated with the stream to the underlying medium.

__close

Close the stream.

CTL_STREAM_NULL

Synopsis

```
#define CTL_STREAM_NULL             (-1)
```

Description

CTL_STREAM_NULL identifies a null stream that is always invalid.

CTL_STREAM_STANDARD_DEBUG

Synopsis

```
#define CTL_STREAM_STANDARD_DEBUG   3
```

Description

CTL_STREAM_STANDARD_DEBUG is a distinguished stream handle that is an alias for semi-hosted input and output.

CTL_STREAM_STANDARD_ERROR

Synopsis

```
#define CTL_STREAM_STANDARD_ERROR   2
```

Description

CTL_STREAM_STANDARD_ERROR is a distinguished stream handle that is an alias for the standard error stream of the executing task.

CTL_STREAM_STANDARD_INPUT

Synopsis

```
#define CTL_STREAM_STANDARD_INPUT   0
```

Description

CTL_STREAM_STANDARD_INPUT is a distinguished stream handle that is an alias for the standard input stream of the executing task.

CTL_STREAM_STANDARD_OUTPUT

Synopsis

```
#define CTL_STREAM_STANDARD_OUTPUT  1
```

Description

CTL_STREAM_STANDARD_OUTPUT is a distinguished stream handle that is an alias for the standard output stream of the executing task.

CTL_STREAM_t

Synopsis

```
typedef long CTL_STREAM_t;
```

Description

CTL_STREAM_t is a generic stream handle.

ctl_atomic_decrement

Synopsis

```
long ctl_atomic_decrement(long *p);
```

Description

ctl_atomic_decrement decrements the long pointed to by **p** and returns the new value of the object. The decrement is performed with global interrupts disabled.

Return Value

The value of the newly decremented object.

ctl_atomic_increment

Synopsis

```
long ctl_atomic_increment(long *p);
```

Description

ctl_atomic_increment increments the long pointed to by **p** and returns the new value of the object. The increment is performed with global interrupts disabled.

Return Value

The value of the newly incremented object.

ctl_byte_swap_16b

Synopsis

```
unsigned ctl_byte_swap_16b(unsigned short n);
```

Description

ctl_byte_swap_16b reverses the byte order of the least significant 16 bits of **n** and returns that value.

See Also

ctl_byte_swap_32b, CTL_BYTE_SWAP_16b_LIT

ctl_byte_swap_32b

Synopsis

```
unsigned long ctl_byte_swap_32b(unsigned long n);
```

Description

ctl_byte_swap_32b reverses the byte order of the four bytes of **n** and returns that value.

See Also

ctl_byte_swap_16b, CTL_BYTE_SWAP_32b_LIT

ctl_delay

Synopsis

```
void ctl_delay(CTL_TIME_t t);
```

Description

ctl_delay is a convenience function that wraps **ctl_timeout_wait** to delay task execution for at least **t** CTL ticks.

ctl_load_16b_be

Synopsis

```
unsigned ctl_load_16b_be(unsigned char *data);
```

Description

ctl_load_16b_be reads a 16-bit data item stored as two consecutive bytes in big-endian byte order and pointed to by **data**, and returns that value. **data** can have any byte alignment.

ctl_load_16b_le

Synopsis

```
unsigned ctl_load_16b_le(unsigned char *data);
```

Description

ctl_load_16b_le reads a 16-bit data item stored as two consecutive bytes in little-endian byte order and pointed to by **data**, and returns that value. **data** can have any byte alignment.

ctl_load_24b_be

Synopsis

```
unsigned long ctl_load_24b_be(unsigned char *data);
```

Description

ctl_load_24b_be reads a 24-bit data item stored as three consecutive bytes in big-endian byte order and pointed to by **data**, and returns that value. **data** can have any byte alignment.

ctl_load_24b_le

Synopsis

```
unsigned long ctl_load_24b_le(unsigned char *data);
```

Description

ctl_load_24b_le reads a 24-bit data item stored as three consecutive bytes in little-endian byte order and pointed to by **data**, and returns that value. **data** can have any byte alignment.

CrossWorks Tools Library

ctl_load_32b_be

Synopsis

```
unsigned long ctl_load_32b_be(unsigned char *data);
```

Description

ctl_load_32b_be reads a 32-bit data item stored as four consecutive bytes in big-endian byte order and pointed to by **data**, and returns that value. **data** can have any byte alignment.

ctl_load_32b_le

Synopsis

```
unsigned long ctl_load_32b_le(unsigned char *data);
```

Description

ctl_load_32b_le reads a 32-bit data item stored as four consecutive bytes in little-endian byte order and pointed to by **data**, and returns that value. **data** can have any byte alignment.

ctl_normalize_status

Synopsis

```
CTL_STATUS_t ctl_normalize_status(CTL_STATUS_t stat);
```

Description

ctl_normalize_status normalizes the status **stat**. If **stat** is non-negative, **ctl_normalize_status** returns **CTL_NO_ERROR**, otherwise it returns **stat** unchanged. You can use this to normalize non-zero "additional-status" return values from functions that indicate success with additional information into the standard **CTL_NO_ERROR** status.

See Also

CTL_STATUS_t

ctl_print_status

Synopsis

```
void ctl_print_status(CTL_STATUS_t status);
```

Description

ctl_print_status prints the status **status** to the console. If **status** is positive indicating a 'no error' status with additional information, **this** prints 'success' where *x* is the value of **status** in decimal. If **status** is zero, **ctl_print_status** prints the decoded error.

Thread Safety

ctl_print_status is thread-safe.

ctl_printf

Synopsis

```
int ctl_printf(const char *fmt,
               ...);
```

Description

ctl_printf writes to the standard output stream under control of the string pointed to by **fmt** that specifies how subsequent arguments are converted for output.

If there are insufficient arguments for the format, the behavior is undefined. If the format is exhausted while arguments remain, the excess arguments are evaluated but are otherwise ignored.

Return Value

ctl_printf returns the number of characters transmitted, or a negative value if an output or encoding error occurred.

ctl_puts

Synopsis

```
int ctl_puts(const char *text);
```

Description

ctl_puts writes the string pointed to by **text** to the standard output stream without encoding.

Return Value

ctl_puts returns the number of characters transmitted, or a negative value if an output error occurred.

Note

ctl_puts differs from the standard function **puts** in that no newline is written to the standard output after **text**.

ctl_register_core_error_decoder

Synopsis

```
void ctl_register_core_error_decoder(void);
```

Description

ctl_register_core_error_decoder registers an error decoder with the CrossWorks runtime library for the standard core errors defined in **CTL_CORE_ERROR_t**.

See Also

CTL_CORE_ERROR_t

ctl_rwlock_claim_read_lock

Synopsis

```
void ctl_rwlock_claim_read_lock(CTL_RWLOCK_t *rwlock);
```

Description

ctl_rwlock_claim_read_lock acquires a read lock on the read-write lock **rwlock**. **ctl_rwlock_claim_read_lock** will block waiting for any write lock on **rwlock** to be released and a read lock on **rwlock** to become available.

ctl_rwlock_claim_read_lock is equivalent to, and is implemented by, a call to **ctl_rwlock_read_lock** specifying **CTL_TIMEOUT_INFINITE** as the timeout type.

Note

ctl_rwlock_claim_read_lock must not be called from an interrupt service routine.

ctl_rwlock_claim_write_lock

Synopsis

```
void ctl_rwlock_claim_write_lock(CTL_RWLOCK_t *rwlock);
```

Description

ctl_rwlock_claim_write_lock acquires an exclusive write lock on the read-write lock **rwlock**.
ctl_rwlock_claim_write_lock will block waiting for all read and write locks on **rwlock** to be released.

ctl_rwlock_claim_write_lock is equivalent to, and is implemented by, a call to **ctl_rwlock_write_lock** specifying **CTL_TIMEOUT_INFINITE** as the timeout type.

ctl_rwlock_init

Synopsis

```
void ctl_rwlock_init(CTL_RWLOCK_t *rwlock,
                     unsigned max_readers);
```

Description

ctl_rwlock_init initializes the read-write lock **rwlock** for simultaneous access by at most **max_readers** without blocking or for exclusive access by one writer.

Increasing the maximum number of simultaneous readers has no effect on the memory requires to realise the lock, however it does increase the time required to acquire an exclusive write lock using **ctl_rwlock_write_lock** or **ctl_rw_write_lock_claim**.

ctl_rwlock_read_lock

Synopsis

```
unsigned ctl_rwlock_read_lock(CTL_RWLOCK_t *rwlock,
                              CTL_TIMEOUT_t t,
                              CTL_TIME_t timeout);
```

Description

this attempts to acquire a read lock on the read-write lock **rwlock**. **ctl_rwlock_read_lock** will block waiting for exclusive access to **rwlock** to end and for an available read lock.

If a timeout is specified and a read lock cannot be acquired before the timeout, **ctl_rwlock_read_lock** returns zero. If a read lock can be acquired, **ctl_rwlock_read_lock** returns a non-zero result.

Note

ctl_rwlock_read_lock must not be called from an interrupt service routine.

ctl_rwlock_read_lock_uc

Synopsis

```
void ctl_rwlock_read_lock_uc(CTL_RWLOCK_t *rwlock);
```

Description

this attempts to acquire a read lock on the read-write lock **rwlock**. **ctl_rwlock_read_lock_uc** will block waiting for exclusive access to **rwlock** to end and for an available read lock.

Note

ctl_rwlock_read_lock_uc must not be called from an interrupt service routine.

ctl_rwlock_release_read_lock

Synopsis

```
void ctl_rwlock_release_read_lock(CTL_RWLOCK_t *rwlock);
```

Description

ctl_rwlock_release_read_lock releases a read lock held on the read-write lock **rwlock**. It is an error to release a read lock that is not held on a read-write lock.

Note

ctl_rwlock_release_read_lock must not be called from an interrupt service routine.

ctl_rwlock_release_write_lock

Synopsis

```
void ctl_rwlock_release_write_lock(CTL_RWLOCK_t *rwlock);
```

Description

ctl_rwlock_release_write_lock releases the write lock held on the read-write lock **rwlock**. It is an error to release a write lock that is not held on a read-write lock.

Note

ctl_rwlock_release_write_lock must not be called from an interrupt service routine.

ctl_rwlock_write_lock

Synopsis

```
unsigned ctl_rwlock_write_lock(CTL_RWLOCK_t *rwlock,
                               CTL_TIMEOUT_t t,
                               CTL_TIME_t timeout);
```

Description

ctl_rwlock_write_lock attempts to acquire an exclusive write lock on the read-write lock **rwlock**. **ctl_rwlock_write_lock** will block waiting for all read and write locks on **rwlock** to be released.

If a timeout is specified and this write lock cannot be acquired before the timeout, **ctl_rwlock_write_lock** returns zero. If this write lock can be acquired, **ctl_rwlock_write_lock** returns a non-zero result.

ctl_stats_group

Synopsis

```
CTL_STATS_GROUP_t *ctl_stats_group(int index);
```

Description

ctl_stats_group returns a pointer to the statistics group with index **index**. If **index** is an invalid index, **ctl_stats_group** returns zero.

ctl_stats_group_count

Synopsis

```
int ctl_stats_group_count(void);
```

Description

ctl_stats_group_count returns the number of statistics groups that have been registered by **ctl_stats_register**.

ctl_stats_print

Synopsis

```
void ctl_stats_print(CTL_STREAM_t s,
                     const char *group);
```

Description

ctl_stats_print prints the statistics group **group** to the stream **s**.

ctl_stats_register

Synopsis

```
void ctl_stats_register(const char *group,
                        const CTL_STATS_DESCRIPTOR_t *desc);
```

Description

ctl_stats_register registers the array of statistics pointed to by **desc** with the group name **group**. The array of statistics is terminated by a statistics descriptor with a zero **name** member.

Note

At most ten statistics groups can be registered. If you require more statistics groups, modify the source code appropriately.

ctl_stream_alloc

Synopsis

```
CTL_STREAM_t ctl_stream_alloc(CTL_STREAM_CB_t *cb);
```

Description

ctl_stream_alloc allocates a new stream and associates it with the control block **cb**. The `signature` member in **cb** is a client-provided value that it uses to identify the type of stream when a stream is passed to stream-specific driver functions.

Return Value

ctl_stream_alloc returns the newly allocated stream handle.

ctl_stream_assign_standard

Synopsis

```
CTL_STATUS_t ctl_stream_assign_standard(CTL_STREAM_t standard,
                                        CTL_STREAM_t handle);
```

Description

ctl_stream_assign_standard sets the standard stream **standard** to use the stream **handle** for input and output operations. The stream **standard** must be one of the standard streams (input, output, or error).

Return Value

ctl_stream_assign_standard returns a standard status code.

Thread Safety

ctl_stream_assign_standard is thread-safe.

ctl_stream_close

Synopsis

```
CTL_STATUS_t ctl_stream_close(CTL_STREAM_t s);
```

Description

ctl_stream_close closes the stream **s**. The implementation of stream closure is defined by the driver associated with the stream.

Return Value

ctl_stream_close returns a standard status code.

ctl_stream_find_cb

Synopsis

```
CTL_STREAM_CB_t *ctl_stream_find_cb(CTL_STREAM_t s);
```

Description

ctl_stream_find_cb returns the generic stream control block for the stream handle **s**. If **s** is not a valid stream handle, **ctl_stream_find_cb** returns zero.

ctl_stream_flush

Synopsis

```
CTL_STATUS_t ctl_stream_flush(CTL_STREAM_t s);
```

Description

ctl_stream_flush flushes the stream **s**. The implementation of stream flushing is defined by the driver associated with the stream.

Return Value

ctl_stream_flush returns a standard status code.

ctl_stream_free_cb

Synopsis

```
void ctl_stream_free_cb(CTL_STREAM_CB_t *cb);
```

Description

ctl_stream_free_cb free the generic stream control block **cb**.

ctl_stream_getchar

Synopsis

```
CTL_STATUS_t ctl_stream_getchar(CTL_STREAM_t s);
```

Description

ctl_stream_getchar reads one byte from the stream **s** and returns it.

Return Value

ctl_stream_getchar returns the byte read, or a negative value if an error occurred.

ctl_stream_is_null

Synopsis

```
int ctl_stream_is_null(CTL_STREAM_t stream);
```

Description

ctl_stream_is_null returns non-zero if **stream** is a null stream and zero if it is not a null stream.

Note

It is not enough to compare a stream handle to **CTL_STREAM_NULL** as the standard streams (**CTL_STREAM_STANDARD_INPUT** etc) are non-zero stream handled and require unmapping by **ctl_stream_unmap** to indicate whether they are attched to a non-null stream.

Thread Safety

ctl_stream_is_null is thread-safe.

ctl_stream_is_open

Synopsis

```
int ctl_stream_is_open(CTL_STREAM_t stream);
```

Description

ctl_stream_is_open returns non-zero if **stream** is currently open and zero if it is not open.

Thread Safety

ctl_stream_is_open is thread-safe.

ctl_stream_printf

Synopsis

```
int ctl_stream_printf(CTL_STREAM_t s,
                      const char *fmt,
                      ...);
```

Description

ctl_stream_printf writes to the stream **s** under control of the string pointed to by **fmt** that specifies how subsequent arguments are converted for output.

If there are insufficient arguments for the format, the behavior is undefined. If the format is exhausted while arguments remain, the excess arguments are evaluated but are otherwise ignored.

Return Value

ctl_stream_printf returns the number of characters transmitted, or a negative value if an output or encoding error occurred.

ctl_stream_read

Synopsis

```
CTL_STATUS_t ctl_stream_read(CTL_STREAM_t s,
                             void *data,
                             size_t len);
```

Description

ctl_stream_read reads **len** bytes of data from the stream **s** into the object pointed to by **data**.

Return Value

ctl_stream_read returns the number of characters received, or a negative value if an error occurred.

ctl_stream_read_line

Synopsis

```
CTL_STATUS_t ctl_stream_read_line(CTL_STREAM_t s,
                                  char *str,
                                  size_t size);
```

Description

ctl_stream_read_line reads a line from the stream **s** (terminated by a new line character) and writes it into the string pointed to by **str** which is **size** bytes in size. This will terminate early if **size**−1 characters are read from the stream without a new line.

Return Value

ctl_stream_read_line returns a standard status code.

ctl_stream_signature

Synopsis

```
unsigned long ctl_stream_signature(CTL_STREAM_t stream);
```

Description

ctl_stream_signature returns the signature of stream **stream**. If **stream** is not a valid stream, **ctl_stream_signature** returns zero.

Thread Safety

ctl_stream_signature is thread-safe.

ctl_stream_unmap

Synopsis

```
CTL_STREAM_t ctl_stream_unmap(CTL_STREAM_t s);
```

Description

ctl_stream_unmap unmaps one of the standard streams (input, output, and error) to the stream handle to the stream that it is mapped from. For other stream handles, **s** is returned unchanged.

Thread Safety

ctl_stream_unmap is thread-safe.

ctl_stream_vprintf

Synopsis

```
int ctl_stream_vprintf(CTL_STREAM_t s,
                       const char *fmt,
                       va_list ap);
```

Description

ctl_stream_vprintf writes to the stream **s** under control of the string pointed to by **fmt** that specifies how subsequent arguments are converted for output. Before calling **ctl_stream_vprintf**, **ap** must be initialized by the **va_start** macro (and possibly subsequent **va_arg** calls). **ctl_stream_vprintf** does not invoke the **va_end** macro.

Return Value

ctl_stream_vprintf returns the number of characters transmitted, or a negative value if an output or encoding error occurred.

Note

ctl_stream_vprintf is equivalent to **ctl_stream_printf** with the variable argument list replaced by **ap**.

ctl_stream_write

Synopsis

```
CTL_STATUS_t ctl_stream_write(CTL_STREAM_t s,
                              const void *data,
                              size_t len);
```

Description

ctl_stream_write writes **len** bytes of data to the stream **s** from the object pointed to by **data**.

Return Value

ctl_stream_write returns the number of characters transmitted, or a negative value if an error occurred.

ctl_strerror

Synopsis

```
char *ctl_strerror(CTL_STATUS_t code);
```

Description

ctl_strerror decode the error code **code** using error decoders registered with the CrossWorks runtime library.

Note

ctl_strerror differs from the standard **strerror** function in that the return type is `const char *` rather than `char *`.

Thread Safety

ctl_strerror is thread-safe.

ctl_system_memory_allocator

Synopsis

```
CTL_MEMORY_ALLOCATOR_t ctl_system_memory_allocator;
```

Description

ctl_system_memory_allocator defines the memory allocator that many of the CrossWorks libraries use to allocate memory.

ctl_task_decode_state

Synopsis

```
char *ctl_task_decode_state(unsigned state);
```

Description

ctl_task_decode_state returns a pointer to a string describing the running state of the task. You must pass in the **state** member of the **CTL_TASK_t** structure.

ctl_vprintf

Synopsis

```
int ctl_vprintf(const char *fmt,
                va_list ap);
```

Description

ctl_vprintf writes to the standard output stream under control of the string pointed to by **fmt** that specifies how subsequent arguments are converted for output. Before calling **ctl_vprintf**, **ap** must be initialized by the **va_start** macro (and possibly subsequent **va_arg** calls). **ctl_vprintf** does not invoke the **va_end** macro.

Return Value

ctl_vprintf returns the number of characters transmitted, or a negative value if an output error occurred.

platform_spin_delay_ms

Synopsis

```
void platform_spin_delay_ms(unsigned period);
```

Description

platform_spin_delay_ms delays execution by busy-waiting for at least *period* milliseconds. The user is responsible for implementing this, or including the Platform Library into the application which implements this.

platform_spin_delay_us

Synopsis

```
void platform_spin_delay_us(unsigned period);
```

Description

platform_spin_delay_us delays execution by busy-waiting for at least *period* microseconds. The user is responsible for implementing this, or including the Platform Library into the application which implements this.

<ctl_heap.h>

Overview

A simple heap implementation.

The utility of this heap implementation lies in the fact that memory can be allocated and freed by tasks, but also that memory can be freed from inside an interrupt handler. The Network Library uses this facility when transmitting packets through the MAC.

API Summary

Heap	
CTL_HEAP_STATISTICS_t	Statistics maintained by heap
CTL_HEAP_t	Heap descriptor
ctl_heap_alloc	Allocate memory
ctl_heap_can_alloc	Test memory availability
ctl_heap_free	Allocate memory
ctl_heap_init	Initialize heap
ctl_heap_trim	Trim memory

CTL_HEAP_STATISTICS_t

Synopsis

```
typedef struct {
  long successMallocCalls;
  long failedMallocCalls;
  long freeCalls;
  long trimCalls;
  long noopTrimCalls;
  long allocCount;
  long blockCount;
  long largestFreeBlockSize;
  long allocCountHighWater;
  long blockCountHighWater;
  long largestFreeBlockSizeLowWater;
  long allFreeMallocHitCount;
  long heap_size;
} CTL_HEAP_STATISTICS_t;
```

Description

CTL_HEAP_STATISTICS_t contains statistics maintained by the heap.

Structure

successMallocCalls

Number of calls to **ctl_heap_alloc** that were satisfied with a memory block.

failedMallocCalls

Number of calls to **ctl_heap_alloc** where allocation failed to return a memory block.

freeCalls

Number of calls to **ctl_heap_free** returning a non-zero memory block.

trimCalls

Number of calls to **ctl_heap_trim**.

trimCalls

Number of calls to **ctl_heap_trim** where there was nothing to do.

allocCount

Number of allocated memory blocks in the heap.

blockCount

Total number of memory blocks in the heap.

largestFreeBlockSize

The number of bytes in the largest free block.

allocCountHighWater

The highest recorded number of allocated memory blocks.

allocCountHighWater

The highest recorded number of total memory blocks.

largestFreeBlockSizeLowWater

The smallest recorded free block size.

allFreeMallocHitCount

The number of times **ctl_heap_alloc** entered with all blocks free.

heap_size

The size of the heap, in bytes.

CTL_HEAP_t

Synopsis

```
typedef struct {
  unsigned long *first;
  unsigned long wordSize;
  unsigned long *lastPlusOne;
  CTL_MUTEX_t allocLock;
  CTL_HEAP_STATISTICS_t stats;
} CTL_HEAP_t;
```

Description

CTL_HEAP_t describes the memory area maintained as a heap.

Structure

`first`

Pointer to the first word in the heap.

`wordSize`

The size, in words, of the heap.

`lastPlusOne`

A pointer to one word beyond the heap, i.e. the first word that is not in the heap.

`allocLock`

A mutex to prevent simultaneous heap allocations.

`stats`

Statistics maintained to monitor the performance of the heap.

ctl_heap_alloc

Synopsis

```
void *ctl_heap_alloc(CTL_HEAP_t *self,
                     size_t count);
```

Description

ctl_heap_alloc allocates **count** bytes from the heap **self** and returns a pointer to the start of the memory block. If **count** bytes cannot be allocated, **this** returns zero.

ctl_heap_can_alloc

Synopsis

```
unsigned ctl_heap_can_alloc(CTL_HEAP_t *heap,
                            size_t byteSize,
                            unsigned nBlocks);
```

Description

ctl_heap_can_alloc tests to see in **nBlocks** of size **byteSize** can be allocated in the heap **self**. Note that success is not an indication that a subsequent allocation will also succeed as other tasks may allocation memory from the heap. In this case, lock the heap mutex, run the test, and if successful, allocate the memory, and finally free the mutex.

ctl_heap_free

Synopsis

```
void ctl_heap_free(void *p);
```

Description

ctl_heap_free frees the memory allocated to the block **p**. If **p** is zero, **ctl_heap_free** does nothing.

Note

The implementation of **ctl_heap_free** allows memory to be freed from an interrupt handler.

ctl_heap_init

Synopsis

```
void ctl_heap_init(CTL_HEAP_t *self,
                   void *mem,
                   size_t size);
```

Description

ctl_heap_init initializes the heap **self** to use the memory pointed to by **mem**, which must be word aligned, of **size** bytes as a heap. The heap statistics are zeroed.

ctl_heap_trim

Synopsis

```
void ctl_heap_trim(CTL_HEAP_t *self,
                   void *p,
                   size_t count);
```

Description

ctl_heap_trim trims the previously-allocated memory block **p** in the heap **self** to **count** bytes. If the previously-allocated memory block is not greater than **count** bytes in size, the trim request is ignored.

<ctl_cli.h>

Overview

Small command line interpreter API.

API Summary

Types	
CTL_CLI_COMMAND_FUNCTION_t	CLI command function
CTL_CLI_CONTEXT_t	CLI instance
CTL_CLI_KEY_t	Compressed keyboard codes
CLI	
ctl_cli_erase_line	Erase current line
ctl_cli_execute	Execute a command line
ctl_cli_find_core_command	Find a core command
ctl_cli_getchar	Get one character
ctl_cli_gets	Line-oriented CLI input
ctl_cli_init	Initialize CLI instance
ctl_cli_read_eval_print	Read-eval-print loop
ctl_cli_replace_input	Replace current line

CTL_CLI_COMMAND_FUNCTION_t

Synopsis

```
typedef void (*CTL_CLI_COMMAND_FUNCTION_t)(void);
```

Description

CTL_CLI_COMMAND_FUNCTION_t defines a `void` function that executes a CLI command.

CTL_CLI_CONTEXT_t

Synopsis

```
typedef struct {
  unsigned terminal_width;
  unsigned terminal_height;
  size_t cursor;
  char *buf;
  int history_cursor;
  int (*process_key)(CTL_CLI_CONTEXT_t *, int);
  void (*process_interrupt)(void);
  CTL_CLI_COMMAND_FUNCTION_t (*find_command)(CTL_CLI_CONTEXT_t *, const char *);
  unsigned char telnet_options[];
} CTL_CLI_CONTEXT_t;
```

Description

CTL_CLI_CONTEXT_t defines an instance of the command line interpreter.

Structure

terminal_width

The width of the terminal, in columns. For telnet sessions, where window size is negotiated, this is set by the negotiation.

terminal_height

The height of the terminal, in rows. For telnet sessions, where window size is negotiated, this is set by the negotiation.

cursor

Private position of the cursor when editing a line.

buf

Pointer to the string to use when reading lines from the user.

history_cursor

The position within the command line history. This is maintained outside of the CLI methods.

process_key

Process a keyboard input.

process_interrupt

Process an interrupt request from the user (hitting the interrupt key, rather than a hardware interrupt request).

find_command

Search for a command and return the processing function.

telnet_options

Private data holding the telnet option state.

CTL_CLI_KEY_t

Synopsis

```
typedef enum {
  CLI_MOUSE_EVENT,
  CURSOR_LEFT,
  CURSOR_RIGHT,
  CURSOR_UP,
  CURSOR_DOWN,
  CURSOR_HOME,
  CURSOR_END,
  PAGE_UP,
  PAGE_DOWN,
  KEY_F1,
  KEY_F2,
  KEY_F3,
  KEY_F4,
  KEY_F5,
  KEY_F6,
  KEY_F7,
  KEY_F8,
  KEY_F9,
  KEY_F10,
  KEY_F11,
  KEY_F12
} CTL_CLI_KEY_t;
```

Description

CTL_CLI_KEY_t defines the compressed keyboard codes that are delivered by **ctl_cli_getchar**.

See Also

ctl_cli_getchar

ctl_cli_erase_line

Synopsis

```
void ctl_cli_erase_line(CTL_CLI_CONTEXT_t *self);
```

Description

ctl_cli_erase_line clears the input buffer of **self** during a call to **ctl_cli_gets**. Typically this method is invoked as part of the CLI's **process_key** method when it detects a special key combination.

ctl_cli_execute

Synopsis

```
void ctl_cli_execute(CTL_CLI_CONTEXT_t *self,
                     char *text);
```

Description

ctl_cli_execute executes the CLI command in **text**. Note that **text** is a non-const pointer and will be modified by **ctl_cli_execute** as it parses the command line.

ctl_cli_find_core_command

Synopsis

```
CTL_CLI_COMMAND_FUNCTION_t ctl_cli_find_core_command(const char *str);
```

Description

ctl_cli_find_core_command searches for the command **str** and returns a non-null function pointer if it is a core command.

Currently the core commands are:

- `stats`
- `streams`
- `date`

Thread Safety

ctl_cli_find_core_command is thread-safe.

ctl_cli_getchar

Synopsis

```
CTL_STATUS_t ctl_cli_getchar(CTL_CLI_CONTEXT_t *self);
```

Description

ctl_cli_getchar waits for and returns a single character for the the CLI **self**. **ctl_cli_getchar** processes standard NVT and ANSI sequences and returns these sequences as single values; for example, the three-character sequence "Escape, [, A" is recognized as a cursor up sequence and is compressed to the single CURSOR_UP code.

ctl_cli_gets

Synopsis

```
CTL_STATUS_t ctl_cli_gets(CTL_CLI_CONTEXT_t *self,
                          char *buf,
                          size_t len,
                          size_t cursor);
```

Description

ctl_cli_gets must be implemented by the user to read a line of text to process by the CLI. If **ctl_cli_gets** returns an error code, the read-eval-print loop is terminated.

Thread Safety

ctl_cli_gets must be implemented to be thread-safe.

ctl_cli_init

Synopsis

```
void ctl_cli_init(CTL_CLI_CONTEXT_t *self);
```

Description

ctl_cli_init initializes an instance of the CrossWorks Console using **area** for management information.

Thread Safety

ctl_cli_init is thread-safe.

ctl_cli_read_eval_print

Synopsis

```
void ctl_cli_read_eval_print(CTL_CLI_CONTEXT_t *self);
```

Description

ctl_cli_read_eval_print starts a CLI read-evaluate-print loop. The only way for **ctl_cli_read_eval_print** to return is for the user to issue the command bye from the console.

Thread Safety

ctl_cli_read_eval_print is thread-safe.

ctl_cli_replace_input

Synopsis

```
void ctl_cli_replace_input(CTL_CLI_CONTEXT_t *self,
                           const char *text);
```

Description

ctl_cli_replace_input replaces the input buffer of **self** with the string pointed to by **text**. Typically this method is invoked as part of the CLI's **process_key** method when it detects a special key combination to scroll through command line history.

<ctl_memchk.h>

Overview

A collection of memory tests.

Traditional tests

- *Checkerboard*:
- *MSCAN*:
- *MOVI*:
- *Zero-One test*:

Marching Tests

A March test consists of a sequence of March steps, while a March step is a finite sequence of operations applied to every cell in the memory array before proceeding to the next cell. An operation is composed of four operations:

- *w0*: write zero into a cell
- *w1*: write one into a cell
- *r0*: read an expected zero from a cell
- *r1*: read an expected one from a cell.

These steps may be ordered by address (increasing address order, decreasing address order), or unordered (where the order doesn't matter).

The implementations in this library are for word-oriented memory. Each implementation is provided in three bus widths (8, 16, and 32 bits wide). To test your RAM, you should run the test that matches your data bus width exactly. Running tests with other bus widths will uncover problems with byte-lane selects (where the tested width is narrower than the actual bus width) or split-writes (where the tested width is wider than the actual bus width).

API Summary

Checkerboard	
ctl_memchk_checkerboard_test_16b	Checkerboard test over memory (16-bit)
ctl_memchk_checkerboard_test_32b	Checkerboard test over memory (32-bit)
ctl_memchk_checkerboard_test_8b	Checkerboard test over memory (8-bit)
MSCAN	
ctl_memchk_mscan_test_16b	MSCAN test (16-bit)
ctl_memchk_mscan_test_32b	MSCAN test (32-bit)
ctl_memchk_mscan_test_8b	MSCAN test (8-bit)

MOVI	
ctl_memchk_movi_test_16b	MOVI test (16-bit)
ctl_memchk_movi_test_32b	MOVI test (32-bit)
ctl_memchk_movi_test_8b	MOVI test (8-bit)
Marching One-Zero	
ctl_memchk_marching_one_zero_test_16b	Marching one-zero test (16-bit)
ctl_memchk_marching_one_zero_test_32b	Marching one-zero test (32-bit)
ctl_memchk_marching_one_zero_test_8b	Marching one-zero test (8-bit)
MATS+	
ctl_memchk_mats_plus_test_16b	MATS+ test (16-bit)
ctl_memchk_mats_plus_test_32b	MATS+ test (32-bit)
ctl_memchk_mats_plus_test_8b	MATS+ test (8-bit)
MATS++	
ctl_memchk_mats_plus_plus_test_16b	MATS++ test (16-bit)
ctl_memchk_mats_plus_plus_test_32b	MATS++ test (32-bit)
ctl_memchk_mats_plus_plus_test_8b	MATS++ test (8-bit)
MARCH C	
ctl_memchk_march_c_test_16b	MARCH C test (16-bit)
ctl_memchk_march_c_test_32b	MARCH C test (32-bit)
ctl_memchk_march_c_test_8b	MARCH C test (8-bit)
MARCH C-	
ctl_memchk_march_c_minus_test_16b	MARCH C- test (16-bit)
ctl_memchk_march_c_minus_test_32b	MARCH C- test (32-bit)
ctl_memchk_march_c_minus_test_8b	MARCH C- test (8-bit)
Extended MARCH C-	
ctl_memchk_extended_march_c_minus_test_16b	Extended MARCH C- test (16-bit)
ctl_memchk_extended_march_c_minus_test_32b	Extended MARCH C- test (32-bit)
ctl_memchk_extended_march_c_minus_test_8b	Extended MARCH C- test (8-bit)
MARCH A	
ctl_memchk_march_a_test_16b	MARCH A test (16-bit)
ctl_memchk_march_a_test_32b	MARCH A test (32-bit)
ctl_memchk_march_a_test_8b	MARCH A test (8-bit)
MARCH B	
ctl_memchk_march_b_test_16b	MARCH B test (16-bit)
ctl_memchk_march_b_test_32b	MARCH B test (32-bit)

ctl_memchk_march_b_test_8b	MARCH B test (8-bit)
MARCH X	
ctl_memchk_march_x_test_16b	MARCH X test (16-bit)
ctl_memchk_march_x_test_32b	MARCH X test (32-bit)
ctl_memchk_march_x_test_8b	MARCH X test (8-bit)
MARCH Y	
ctl_memchk_march_y_test_16b	MARCH Y test (16-bit)
ctl_memchk_march_y_test_32b	MARCH Y test (32-bit)
ctl_memchk_march_y_test_8b	MARCH Y test (8-bit)
MARCH LR	
ctl_memchk_march_lr_test_16b	MARCH LR test (16-bit)
ctl_memchk_march_lr_test_32b	MARCH LR test (32-bit)
ctl_memchk_march_lr_test_8b	MARCH LR test (8-bit)
MARCH M	
ctl_memchk_march_m_test_16b	MARCH M test (16-bit)
ctl_memchk_march_m_test_32b	MARCH M test (32-bit)
ctl_memchk_march_m_test_8b	MARCH M test (8-bit)
MARCH M--OR	
ctl_memchk_march_m_minus_minus_or_test_16b	MARCH M--OR test (16-bit)
ctl_memchk_march_m_minus_minus_or_test_32b	MARCH M--OR test (32-bit)
ctl_memchk_march_m_minus_minus_or_test_8b	MARCH M--OR test (8-bit)
MARCH M--AND	
ctl_memchk_march_m_minus_minus_and_test_16b	MARCH M--AND test (16-bit)
ctl_memchk_march_m_minus_minus_and_test_32b	MARCH M--AND test (32-bit)
ctl_memchk_march_m_minus_minus_and_test_8b	MARCH M--AND test (8-bit)
MARCH G	
ctl_memchk_march_g_test_16b	MARCH G test (16-bit)
ctl_memchk_march_g_test_32b	MARCH G test (32-bit)
ctl_memchk_march_g_test_8b	MARCH G test (8-bit)
Symmetric MARCH G	
ctl_memchk_symmetric_march_g_test_16b	Symmetric MARCH G test (16-bit)
ctl_memchk_symmetric_march_g_test_32b	Symmetric MARCH G test (32-bit)
ctl_memchk_symmetric_march_g_test_8b	Symmetric MARCH G test (8-bit)
IFA-9	
ctl_memchk_ifa_9_test_16b	IFA-9 test (16-bit)

ctl_memchk_ifa_9_test_32b	IFA-9 test (32-bit)
ctl_memchk_ifa_9_test_8b	IFA-9 test (8-bit)
IFA-13	
ctl_memchk_ifa_13_test_16b	IFA-13 test (16-bit)
ctl_memchk_ifa_13_test_32b	IFA-13 test (32-bit)
ctl_memchk_ifa_13_test_8b	IFA-13 test (8-bit)

ctl_memchk_checkerboard_test_16b

Synopsis

```
CTL_STATUS_t ctl_memchk_checkerboard_test_16b(void *start,
                                              size_t byte_count);
```

Description

As ctl_memchk_checkerboard_test_8b but uses 16-bit units.

ctl_memchk_checkerboard_test_32b

Synopsis

```
CTL_STATUS_t ctl_memchk_checkerboard_test_32b(void *start,
                                              size_t byte_count);
```

Description

As ctl_memchk_checkerboard_test_8b but uses 32-bit units.

ctl_memchk_checkerboard_test_8b

Synopsis

```
CTL_STATUS_t ctl_memchk_checkerboard_test_8b(void *start,
                                             size_t byte_count);
```

Description

ctl_memchk_checkerboard_test_8b runs a checkerboard test over the memory pointed to by **start** of **byte_count** bytes.

Return Value

ctl_memchk_checkerboard_test_8b returns a standard status code.

References

M.A. Breuer and A. D. Friedman, "Diagnosis and Reliable Design of Digital Systems," Computer Science Press, 1976.

ctl_memchk_extended_march_c_minus_test_16b

Synopsis

```
CTL_STATUS_t ctl_memchk_extended_march_c_minus_test_16b(void *start,
                                                        size_t byte_count);
```

Description

As ctl_memchk_extended_march_c_minus_test_8b but uses 16-bit units.

ctl_memchk_extended_march_c_minus_test_32b

Synopsis

```
CTL_STATUS_t ctl_memchk_extended_march_c_minus_test_32b(void *start,
                                                        size_t byte_count);
```

Description

As ctl_memchk_extended_march_c_minus_test_8b but uses 32-bit units.

ctl_memchk_extended_march_c_minus_test_8b

Synopsis

```
CTL_STATUS_t ctl_memchk_extended_march_c_minus_test_8b(void *start,
                                                       size_t byte_count);
```

Description

ctl_memchk_extended_march_c_minus_test_8b runs an Extended MARCH C- test over the memory pointed to by **start** of **byte_count** bytes.

The marching steps are:

?(w0); #(r0,w1,r1); #(r1,w0);
#(r0,w1); #(r1,w0); ?(r0)

Return Value

ctl_memchk_extended_march_c_minus_test_8b returns a standard status code.

ctl_memchk_ifa_13_test_16b

Synopsis

```
CTL_STATUS_t ctl_memchk_ifa_13_test_16b(void *start,
                                        size_t byte_count,
                                        CTL_MEMCHK_DEL_FUNCTION_t del);
```

Description

As ctl_memchk_ifa_13_test_8b but uses 16-bit units.

ctl_memchk_ifa_13_test_32b

Synopsis

```
CTL_STATUS_t ctl_memchk_ifa_13_test_32b(void *start,
                                        size_t byte_count,
                                        CTL_MEMCHK_DEL_FUNCTION_t del);
```

Description

As ctl_memchk_ifa_13_test_8b but uses 32-bit units.

ctl_memchk_ifa_13_test_8b

Synopsis

```
CTL_STATUS_t ctl_memchk_ifa_13_test_8b(void *start,
                                       size_t byte_count,
                                       CTL_MEMCHK_DEL_FUNCTION_t del);
```

Description

ctl_memchk_ifa_13_test_8b runs an IFA-13 test over the memory pointed to by **start** of **byte_count** bytes.

The marching steps are:

?(w0); #(r0,w1,r1); #(r1,w0,r0);
#(r0,w1,r1); #(r1,w0,r0);
del;
#(r0,w1);
del;
#(r1)

Return Value

ctl_memchk_ifa_13_test_8b returns a standard status code.

References

R. Dekker, F. Beenker, L. Thijssen, "A Realistic Fault Model and Test Algorithms for Static Random Access Memories," IEEE Transactions on Computer-Aided Design, Vol 9, No 6.

ctl_memchk_ifa_9_test_16b

Synopsis

```
CTL_STATUS_t ctl_memchk_ifa_9_test_16b(void *start,
                                       size_t byte_count,
                                       CTL_MEMCHK_DEL_FUNCTION_t del);
```

Description

As ctl_memchk_ifa_9_test_8b but uses 16-bit units.

ctl_memchk_ifa_9_test_32b

Synopsis

```
CTL_STATUS_t ctl_memchk_ifa_9_test_32b(void *start,
                                       size_t byte_count,
                                       CTL_MEMCHK_DEL_FUNCTION_t del);
```

Description

As ctl_memchk_ifa_9_test_8b but uses 32-bit units.

ctl_memchk_ifa_9_test_8b

Synopsis

```
CTL_STATUS_t ctl_memchk_ifa_9_test_8b(void *start,
                                      size_t byte_count,
                                      CTL_MEMCHK_DEL_FUNCTION_t del);
```

Description

ctl_memchk_ifa_9_test_8b runs an IFA-9 test over the memory pointed to by **start** of **byte_count** bytes.

The marching steps are:

?(w0); #(r0,w1); #(r1,w0);
#(r0,w1); #(r1,w0);
del;
#(r0,w1);
del;
#(r1)

Return Value

ctl_memchk_ifa_9_test_8b returns a standard status code.

References

R. Dekker, F. Beenker, L. Thijssen, "A Realistic Fault Model and Test Algorithms for Static Random Access Memories," IEEE Transactions on Computer-Aided Design, Vol 9, No 6.

ctl_memchk_march_a_test_16b

Synopsis

```
CTL_STATUS_t ctl_memchk_march_a_test_16b(void *start,
                                         size_t byte_count);
```

Description

As ctl_memchk_march_a_test_8b but uses 16-bit units.

ctl_memchk_march_a_test_32b

Synopsis

```
CTL_STATUS_t ctl_memchk_march_a_test_32b(void *start,
                                         size_t byte_count);
```

Description

As ctl_memchk_march_a_test_8b but uses 32-bit units.

ctl_memchk_march_a_test_8b

Synopsis

```
CTL_STATUS_t ctl_memchk_march_a_test_8b(void *start,
                                        size_t byte_count);
```

Description

ctl_memchk_march_a_test_8b runs a MARCH A test over the memory pointed to by **start** of **byte_count** bytes.

The marching steps are:

?(w0); #(r0,w1,w0,w1); #(r1,w0,w1);
#(r1,w0,w1,w0); #(r0,w1,w0)

Return Value

ctl_memchk_march_a_test_8b returns a standard status code.

References

D. S. Suk and S. M. Reddy, "A March Test for Functional Faults in Semiconductor Random-Access Memories," IEEE Trans. Computers, Vol. C-30, No. 12, 1981.

ctl_memchk_march_b_test_16b

Synopsis

```
CTL_STATUS_t ctl_memchk_march_b_test_16b(void *start,
                                         size_t byte_count);
```

Description

As ctl_memchk_march_b_test_8b but uses 16-bit units.

ctl_memchk_march_b_test_32b

Synopsis

```
CTL_STATUS_t ctl_memchk_march_b_test_32b(void *start,
                                         size_t byte_count);
```

Description

As ctl_memchk_march_b_test_8b but uses 32-bit units.

ctl_memchk_march_b_test_8b

Synopsis

```
CTL_STATUS_t ctl_memchk_march_b_test_8b(void *start,
                                        size_t byte_count);
```

Description

ctl_memchk_march_b_test_8b runs a MARCH B test over the memory pointed to by **start** of **byte_count** bytes.

The marching steps are:

?(w0); #(r0,w1,r1,w0,r0,w1); #(r1,w0,w1);
#(r1,w0,w1,w0); #(r0,w1,w0)

Return Value

ctl_memchk_march_b_test_8b returns a standard status code.

References

D. S. Suk and S. M. Reddy, "A March Test for Functional Faults in Semiconductor Random-Access Memories," IEEE Trans. Computers, Vol. C-30, No. 12, 1981.

ctl_memchk_march_c_minus_test_16b

Synopsis

```
CTL_STATUS_t ctl_memchk_march_c_minus_test_16b(void *start,
                                               size_t byte_count);
```

Description

As ctl_memchk_march_c_minus_test_8b but uses 16-bit units.

ctl_memchk_march_c_minus_test_32b

Synopsis

```
CTL_STATUS_t ctl_memchk_march_c_minus_test_32b(void *start,
                                               size_t byte_count);
```

Description

As ctl_memchk_march_c_minus_test_8b but uses 32-bit units.

ctl_memchk_march_c_minus_test_8b

Synopsis

```
CTL_STATUS_t ctl_memchk_march_c_minus_test_8b(void *start,
                                              size_t byte_count);
```

Description

ctl_memchk_march_c_minus_test_8b runs a MARCH C- test over the memory pointed to by **start** of **byte_count** bytes.

The marching steps are:

?(w0); #(r0,w1); #(r1,w0);
#(r0,w1); #(r1,w0); ?(r0)

Return Value

ctl_memchk_march_c_minus_test_8b returns a standard status code.

References

A. J. Van De Goor, "Using MARCH tests to test SRAMs," Design and Test of Computers, IEEE Volume 10, Issue 1, March 1993.

ctl_memchk_march_c_test_16b

Synopsis

```
CTL_STATUS_t ctl_memchk_march_c_test_16b(void *start,
                                         size_t byte_count);
```

Description

As ctl_memchk_march_c_test_8b but uses 16-bit units.

ctl_memchk_march_c_test_32b

Synopsis

```
CTL_STATUS_t ctl_memchk_march_c_test_32b(void *start,
                                         size_t byte_count);
```

Description

As ctl_memchk_march_c_test_8b but uses 32-bit units.

ctl_memchk_march_c_test_8b

Synopsis

```
CTL_STATUS_t ctl_memchk_march_c_test_8b(void *start,
                                        size_t byte_count);
```

Description

ctl_memchk_march_c_test_8b runs a MARCH C test over the memory pointed to by **start** of **byte_count** bytes.

The marching steps are:

?(w0); #(r0,w1); #(r1,w0);
?(r0); #(r0,w1); #(r1,w0); ?(r0)

Return Value

ctl_memchk_march_c_test_8b returns a standard status code.

References

Cheng-Wen Wu, "RAM Fault models and Memory Testing," Lab for Reliable Computing (LaRC), NTHU.

ctl_memchk_march_g_test_16b

Synopsis

```
CTL_STATUS_t ctl_memchk_march_g_test_16b(void *start,
                                         size_t byte_count,
                                         CTL_MEMCHK_DEL_FUNCTION_t del);
```

Description

As ctl_memchk_march_g_test_8b but uses 16-bit units.

ctl_memchk_march_g_test_32b

Synopsis

```
CTL_STATUS_t ctl_memchk_march_g_test_32b(void *start,
                                         size_t byte_count,
                                         CTL_MEMCHK_DEL_FUNCTION_t del);
```

Description

As ctl_memchk_march_g_test_8b but uses 32-bit units.

ctl_memchk_march_g_test_8b

Synopsis

```
CTL_STATUS_t ctl_memchk_march_g_test_8b(void *start,
                                        size_t byte_count,
                                        CTL_MEMCHK_DEL_FUNCTION_t del);
```

Description

ctl_memchk_march_g_test_8b runs a MARCH G test over the memory pointed to by **start** of **byte_count** bytes.

The marching steps are:

?(w0); #(r0,w1,r1,w0,r0,w1); #(r1,w0,w1);
#(r1,w0,w1,w0); #(r0,w1,w0);
del;
#(r0,w1,r1);
del;
#(r1,w0,r0)

Return Value

ctl_memchk_march_g_test_8b returns a standard status code.

References

V.G. Mikitjuk, V.N. Yarmolik, A.J. van de Goor, "RAM Testing Algorithms for Detection Multiple Linked Faults," Electronic Design and Test Conference, 1996.

ctl_memchk_march_lr_test_16b

Synopsis

```
CTL_STATUS_t ctl_memchk_march_lr_test_16b(void *start,
                                          size_t byte_count);
```

Description

As ctl_memchk_march_lr_test_8b but uses 16-bit units.

ctl_memchk_march_lr_test_32b

Synopsis

```
CTL_STATUS_t ctl_memchk_march_lr_test_32b(void *start,
                                          size_t byte_count);
```

Description

As ctl_memchk_march_lr_test_8b but uses 32-bit units.

ctl_memchk_march_lr_test_8b

Synopsis

```
CTL_STATUS_t ctl_memchk_march_lr_test_8b(void *start,
                                         size_t byte_count);
```

Description

ctl_memchk_march_lr_test_8b runs a MARCH LR test over the memory pointed to by **start** of **byte_count** bytes.

The marching steps are:

#(w0); #(r0,w1); #(r1,w0,r0,w1);
#(r1,w0); #(r0,w1,r1,w0);
#(r0)

Return Value

ctl_memchk_march_lr_test_8b returns a standard status code.

References

A. J. Van De Goor, "March LR: A Test for Realistic Linked Faults," 14th VLSI Test Symposium.

ctl_memchk_march_m_minus_minus_and_test_16b

Synopsis

```
CTL_STATUS_t ctl_memchk_march_m_minus_minus_and_test_16b(void *start,
                                                         size_t byte_count);
```

Description

As ctl_memchk_march_m_minus_minus_and_test_8b but uses 16-bit units.

ctl_memchk_march_m_minus_minus_and_test_32b

Synopsis

```
CTL_STATUS_t ctl_memchk_march_m_minus_minus_and_test_32b(void *start,
                                                         size_t byte_count);
```

Description

As ctl_memchk_march_m_minus_minus_and_test_8b but uses 32-bit units.

ctl_memchk_march_m_minus_minus_and_test_8b

Synopsis

```
CTL_STATUS_t ctl_memchk_march_m_minus_minus_and_test_8b(void *start,
                                                        size_t byte_count);
```

Description

ctl_memchk_march_m_minus_minus_and_test_8b runs a MARCH M--AND test over the memory pointed to by **start** of **byte_count** bytes.

The marching steps are:

?(w1); #(r1,w0,r0,w1); ?(r1);
#(r1,w0); ?(r0);
#(r0,w1,r1,w0); ?(r0)

Return Value

ctl_memchk_march_m_minus_minus_and_test_8b returns a standard status code.

References

V.G. Mikitjuk, V.N. Yarmolik, A.J. van de Goor, "RAM Testing Algorithms for Detection Multiple Linked Faults," Electronic Design and Test Conference, 1996.

ctl_memchk_march_m_minus_minus_or_test_16b

Synopsis

```
CTL_STATUS_t ctl_memchk_march_m_minus_minus_or_test_16b(void *start,
                                                        size_t byte_count);
```

Description

As ctl_memchk_march_m_minus_minus_or_test_8b but uses 16-bit units.

ctl_memchk_march_m_minus_minus_or_test_32b

Synopsis

```
CTL_STATUS_t ctl_memchk_march_m_minus_minus_or_test_32b(void *start,
                                                        size_t byte_count);
```

Description

As ctl_memchk_march_m_minus_minus_or_test_8b but uses 32-bit units.

ctl_memchk_march_m_minus_minus_or_test_8b

Synopsis

```
CTL_STATUS_t ctl_memchk_march_m_minus_minus_or_test_8b(void *start,
                                                       size_t byte_count);
```

Description

ctl_memchk_march_m_minus_minus_or_test_8b runs a MARCH M--OR test over the memory pointed to by **start** of **byte_count** bytes.

The marching steps are:

?(w0); #(r0,w1,r1,w0); ?(r0);
#(r0,w1); ?(r1);
#(r1,w0,r0,w1); ?(r1)

Return Value

ctl_memchk_march_m_minus_minus_or_test_8b returns a standard status code.

References

V.G. Mikitjuk, V.N. Yarmolik, A.J. van de Goor, "RAM Testing Algorithms for Detection Multiple Linked Faults," Electronic Design and Test Conference, 1996.

ctl_memchk_march_m_test_16b

Synopsis

```
CTL_STATUS_t ctl_memchk_march_m_test_16b(void *start,
                                         size_t byte_count);
```

Description

As ctl_memchk_march_m_test_8b but uses 16-bit units.

ctl_memchk_march_m_test_32b

Synopsis

```
CTL_STATUS_t ctl_memchk_march_m_test_32b(void *start,
                                         size_t byte_count);
```

Description

As ctl_memchk_march_m_test_8b but uses 32-bit units.

ctl_memchk_march_m_test_8b

Synopsis

```
CTL_STATUS_t ctl_memchk_march_m_test_8b(void *start,
                                        size_t byte_count);
```

Description

ctl_memchk_march_m_test_8b runs a MARCH M test over the memory pointed to by **start** of **byte_count** bytes.

The marching steps are:

?(w0); #(r0,w1,r1,w0); ?(r0);
#(r0,w1); ?(r1);
#(r1,w0,r0,w1); ?(r1);
#(r1,w0)

Return Value

ctl_memchk_march_m_test_8b returns a standard status code.

References

V.G. Mikitjuk, V.N. Yarmolik, A.J. van de Goor, "RAM Testing Algorithms for Detection Multiple Linked Faults," Electronic Design and Test Conference, 1996.

ctl_memchk_march_x_test_16b

Synopsis

```
CTL_STATUS_t ctl_memchk_march_x_test_16b(void *start,
                                         size_t byte_count);
```

Description

As ctl_memchk_march_x_test_8b but uses 16-bit units.

ctl_memchk_march_x_test_32b

Synopsis

```
CTL_STATUS_t ctl_memchk_march_x_test_32b(void *start,
                                         size_t byte_count);
```

Description

As ctl_memchk_march_x_test_8b but uses 32-bit units.

ctl_memchk_march_x_test_8b

Synopsis

```
CTL_STATUS_t ctl_memchk_march_x_test_8b(void *start,
                                        size_t byte_count);
```

Description

ctl_memchk_march_x_test_8b runs a MARCH X test over the memory pointed to by **start** of **byte_count** bytes.

The marching steps are:

?(w0); #(r0,w1); #(r1,w0); ?(r0)

Return Value

ctl_memchk_march_x_test_8b returns a standard status code.

ctl_memchk_march_y_test_16b

Synopsis

```
CTL_STATUS_t ctl_memchk_march_y_test_16b(void *start,
                                         size_t byte_count);
```

Description

As ctl_memchk_march_y_test_8b but uses 16-bit units.

ctl_memchk_march_y_test_32b

Synopsis

```
CTL_STATUS_t ctl_memchk_march_y_test_32b(void *start,
                                         size_t byte_count);
```

Description

As ctl_memchk_march_y_test_8b but uses 32-bit units.

ctl_memchk_march_y_test_8b

Synopsis

```
CTL_STATUS_t ctl_memchk_march_y_test_8b(void *start,
                                        size_t byte_count);
```

Description

ctl_memchk_march_y_test_8b runs a MARCH Y test over the memory pointed to by **start** of **byte_count** bytes.

The marching steps are:

?(w0); #(r0,w1,r1); #(r1,w0,r0); ?(r0)

Return Value

ctl_memchk_march_y_test_8b returns a standard status code.

ctl_memchk_marching_one_zero_test_16b

Synopsis

```
CTL_STATUS_t ctl_memchk_marching_one_zero_test_16b(void *start,
                                                   size_t byte_count);
```

Description

As ctl_memchk_marching_one_zero_test_8b but uses 16-bit units.

ctl_memchk_marching_one_zero_test_32b

Synopsis

```
CTL_STATUS_t ctl_memchk_marching_one_zero_test_32b(void *start,
                                                   size_t byte_count);
```

Description

As ctl_memchk_marching_one_zero_test_8b but uses 32-bit units.

ctl_memchk_marching_one_zero_test_8b

Synopsis

```
CTL_STATUS_t ctl_memchk_marching_one_zero_test_8b(void *start,
                                                  size_t byte_count);
```

Description

ctl_memchk_marching_one_zero_test_8b runs a marching one-zero test over the memory pointed to by **start** of **byte_count** bytes.

The marching steps are:

#(w0); #(r0,w1,r1); #(r1,w0,r0);
#(w1); #(r1,w0,r0); #(r,w1,r1)

Return Value

ctl_memchk_marching_one_zero_test_8b returns a standard status code.

References

M.A. Breuer and A. D. Friedman, "Diagnosis and Reliable Design of Digital Systems," Computer Science Press, 1976.

ctl_memchk_mats_plus_plus_test_16b

Synopsis

```
CTL_STATUS_t ctl_memchk_mats_plus_plus_test_16b(void *start,
                                                size_t byte_count);
```

Description

As ctl_memchk_mats_plus_plus_test_8b but uses 32-bit units.

ctl_memchk_mats_plus_plus_test_32b

Synopsis

```
CTL_STATUS_t ctl_memchk_mats_plus_plus_test_32b(void *start,
                                                size_t byte_count);
```

Description

As ctl_memchk_mats_plus_plus_test_8b but uses 32-bit units.

ctl_memchk_mats_plus_plus_test_8b

Synopsis

```
CTL_STATUS_t ctl_memchk_mats_plus_plus_test_8b(void *start,
                                               size_t byte_count);
```

Description

ctl_memchk_mats_plus_plus_test_8b runs a MATS++ test over the memory pointed to by **start** of **byte_count** bytes.

The marching steps are:

?(w0); #(r0,w1); #(r1,w0,r0)

Return Value

ctl_memchk_mats_plus_plus_test_8b returns a standard status code.

References

A. J. Van De Goor, "Using MARCH tests to test SRAMs," Design and Test of Computers, IEEE Volume 10, Issue 1, March 1993.

ctl_memchk_mats_plus_test_16b

Synopsis

```
CTL_STATUS_t ctl_memchk_mats_plus_test_16b(void *start,
                                           size_t byte_count);
```

Description

As ctl_memchk_mats_plus_test_8b but uses 16-bit units.

ctl_memchk_mats_plus_test_32b

Synopsis

```
CTL_STATUS_t ctl_memchk_mats_plus_test_32b(void *start,
                                           size_t byte_count);
```

Description

As ctl_memchk_mats_plus_test_8b but uses 32-bit units.

ctl_memchk_mats_plus_test_8b

Synopsis

```
CTL_STATUS_t ctl_memchk_mats_plus_test_8b(void *start,
                                          size_t byte_count);
```

Description

ctl_memchk_mats_plus_test_8b runs a MATS+ test over the memory pointed to by **start** of **byte_count** bytes.

The marching steps are:

?(w0); #(r0,w1); #(r1,w0)

Return Value

ctl_memchk_mats_plus_test_8b returns a standard status code.

References

8. R. Nair et al., "Efficient Algorithms for Testing Semiconductor Random Access Memories," IEEE Trans. Computers, Vol. C-28, No. 3, 1978.

ctl_memchk_movi_test_16b

Synopsis

```
CTL_STATUS_t ctl_memchk_movi_test_16b(void *start,
                                      size_t byte_count);
```

Description

As ctl_memchk_movi_test_8b but uses 16-bit units.

ctl_memchk_movi_test_32b

Synopsis

```
CTL_STATUS_t ctl_memchk_movi_test_32b(void *start,
                                      size_t byte_count);
```

Description

As ctl_memchk_movi_test_8b but uses 32-bit units.

ctl_memchk_movi_test_8b

Synopsis

```
CTL_STATUS_t ctl_memchk_movi_test_8b(void *start,
                                     size_t byte_count);
```

Description

ctl_memchk_movi_test_8b runs an MOVI test (also called a Moving Inversion test) over the memory pointed to by **start** of **byte_count** bytes.

References

J. H. de Jong, "Moving Inversions Test Pattern is Thorough, Yet Speedy," Computer Design, May 1976.

ctl_memchk_mscan_test_16b

Synopsis

```
CTL_STATUS_t ctl_memchk_mscan_test_16b(void *start,
                                       size_t byte_count);
```

Description

As ctl_memchk_mscan_test_8b but uses 16-bit units.

ctl_memchk_mscan_test_32b

Synopsis

```
CTL_STATUS_t ctl_memchk_mscan_test_32b(void *start,
                                       size_t byte_count);
```

Description

As ctl_memchk_mscan_test_8b but uses 32-bit units.

ctl_memchk_mscan_test_8b

Synopsis

```
CTL_STATUS_t ctl_memchk_mscan_test_8b(void *start,
                                      size_t byte_count);
```

Description

ctl_memchk_mscan_test_8b runs an MSCAN test (also called a zero-one test) over the memory pointed to by **start** of **byte_count** bytes.

The marching steps are:

?(w0); ?(r0); ?(w1); ?(r1)

Return Value

ctl_memchk_mscan_test_8b returns a standard status code.

References

M.A. Breuer and A. D. Friedman, "Diagnosis and Reliable Design of Digital Systems," Computer Science Press, 1976.

ctl_memchk_symmetric_march_g_test_16b

Synopsis

```
CTL_STATUS_t ctl_memchk_symmetric_march_g_test_16b(void *start,
                                                   size_t byte_count,
                                                   CTL_MEMCHK_DEL_FUNCTION_t del);
```

Description

As ctl_memchk_symmetric_march_g_test_8b but uses 16-bit units.

ctl_memchk_symmetric_march_g_test_32b

Synopsis

```
CTL_STATUS_t ctl_memchk_symmetric_march_g_test_32b(void *start,
                                                   size_t byte_count,
                                                   CTL_MEMCHK_DEL_FUNCTION_t del);
```

Description

As ctl_memchk_symmetric_march_g_test_8b but uses 32-bit units.

ctl_memchk_symmetric_march_g_test_8b

Synopsis

```
CTL_STATUS_t ctl_memchk_symmetric_march_g_test_8b(void *start,
                                                  size_t byte_count,
                                                  CTL_MEMCHK_DEL_FUNCTION_t del);
```

Description

ctl_memchk_symmetric_march_g_test_8b runs a Symmetric MARCH G test over the memory pointed to by **start** of **byte_count** bytes.

The marching steps are:

?(w0); #(r0,w1,r1,w0,r0,w1); #(r1,w0,r0,w1);
#(r1,w0,r0,w1); #(r1,w0,w1,w0); #(r0,w1,r1,w0);
del;
#(r0,w1,r1);
del;
#(r1,w0,r0)

Return Value

ctl_memchk_symmetric_march_g_test_8b returns a standard status code.

References

V.G. Mikitjuk, V.N. Yarmolik, A.J. van de Goor, "RAM Testing Algorithms for Detection Multiple Linked Faults," Electronic Design and Test Conference, 1996.