W1: Dallas' 1-wire bus

David Fries <David@Fries.net>

W1: Dallas' 1-wire bus

by David Fries

Copyright © 2013

This documentation is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. For more details see the file COPYING in the source distribution of Linux.

Table of Contents

1. W1 API internal to the kernel 1
W1 API internal to the kernel 1
drivers/w1/w1.h 1
drivers/w1/w1.c 8
drivers/w1/w1_family.h 10
drivers/w1/w1_family.c 12
drivers/w1/w1_int.c 14
drivers/w1/w1_netlink.h 16
drivers/w1/w1_io.c 21

Chapter 1. W1 API internal to the kernel

W1 API internal to the kernel

drivers/w1/w1.h

W1 core functions.

Name

struct w1_reg_num — broken out slave device id

Synopsis

```
struct w1_reg_num {
#if defined(__LITTLE_ENDIAN_BITFIELD)
  __u64 family:8;
  __u64 id:48;
  __u64 crc:8;
#elif defined(__BIG_ENDIAN_BITFIELD)
  __u64 crc:8;
  __u64 id:48;
  __u64 family:8;
#else
#error "Please fix <asm/byteorder.h>"
#endif
};
```

Members

family	identifies the type of device
id	along with family is the unique device id
crc	checksum of the other bytes
crc	checksum of the other bytes
id	along with family is the unique device id
family	identifies the type of device

Name

struct w1_slave — holds a single slave device on the bus

Synopsis

```
struct w1_slave {
  struct module * owner;
  unsigned char name[W1_MAXNAMELEN];
  struct list_head w1_slave_entry;
  struct w1_reg_num reg_num;
  atomic_t refcnt;
  int ttl;
  unsigned long flags;
  struct w1_master * master;
  struct w1_family * family;
  void * family_data;
  struct device dev;
};
```

Members

owner	Points to the one wire “wire” kernel module.
name[W1_MAXNAMELEN]	Device id is ascii.
w1_slave_entry	data for the linked list
reg_num	the slave id in binary
refcnt	reference count, delete when 0
ttl	decrement per search this slave isn't found, deatch at 0
flags	bit flags for W1_SLAVE_ACTIVE W1_SLAVE_DETACH
master	bus which this slave is on
family	module for device family type
family_data	pointer for use by the family module
dev	kernel device identifier

Name

struct w1_bus_master — operations available on a bus master

Synopsis

```
struct w1_bus_master {
  void * data;
  u8 (* read_bit) (void *);
  void (* write_bit) (void *, u8);
  u8 (* touch_bit) (void *, u8);
  u8 (* read_byte) (void *);
  void (* write_byte) (void *, u8);
  u8 (* read_block) (void *, u8 *, int);
  void (* write_block) (void *, const u8 *, int);
  u8 (* triplet) (void *, u8);
  u8 (* reset_bus) (void *);
  u8 (* set_pullup) (void *, int);
  void (* search) (void *, struct w1_master *,u8, w1_slave_found_callback);
};
```

Members

data	the first parameter in all the functions below
read_bit	Sample the line level *return* the level read (0 or 1)
write_bit	Sets the line level
touch_bit	the lowest-level function for devices that really support the 1-wire protocol. touch_bit(0) = write-0 cycle touch_bit(1) = write-1 / read cycle *return* the bit read (0 or 1)
read_byte	Reads a bytes. Same as 8 touch_bit(1) calls. *return* the byte read
write_byte	Writes a byte. Same as 8 touch_bit(x) calls.
read_block	Same as a series of `read_byte` calls *return* the number of bytes read
write_block	Same as a series of `write_byte` calls
triplet	Combines two reads and a smart write for ROM searches *return* bit0=Id bit1=comp_id bit2=dir_taken
reset_bus	long write-0 with a read for the presence pulse detection *return* -1=Error, 0=Device present, 1=No device present
set_pullup	Put out a strong pull-up pulse of the specified duration. *return* -1=Error, 0=completed
search	Really nice hardware can handles the different types of ROM search w1_master* is passed to the slave found callback. u8 is search_type, W1_SEARCH or W1_ALARM_SEARCH

Note

read_bit and write_bit are very low level functions and should only be used with hardware that doesn't really support 1-wire operations, like a parallel/serial port. Either define read_bit and write_bit OR define, at minimum, touch_bit and reset_bus.

Name

enum w1_master_flags — bitfields used in w1_master.flags

Synopsis

```
enum w1_master_flags {
  W1_ABORT_SEARCH,
  W1_WARN_MAX_COUNT
};
```

Constants

W1_ABORT_SEARCH abort searching early on shutdown

W1_WARN_MAX_COUNT limit warning when the maximum count is reached

Name

struct w1_master — one per bus master

Synopsis

```
struct w1_master {
  struct list_head w1_master_entry;
  struct module * owner;
  unsigned char name[W1_MAXNAMELEN];
  struct mutex list_mutex;
  struct list_head slist;
  struct list_head async_list;
  int max_slave_count;
  int slave_count;
  unsigned long attempts;
  int slave_ttl;
  int initialized;
  u32 id;
  int search_count;
  u64 search_id;
  atomic_t refcnt;
  void * priv;
  int enable_pullup;
  int pullup_duration;
  long flags;
  struct task_struct * thread;
  struct mutex mutex;
  struct mutex bus_mutex;
  struct device_driver * driver;
  struct device dev;
  struct w1_bus_master * bus_master;
  u32 seq;
};
```

Members

w1_master_entry	master linked list
owner	module owner
name[W1_MAXNAMELEN]	dynamically allocate bus name
list_mutex	protect slist and async_list
slist	linked list of slaves
async_list	linked list of netlink commands to execute
max_slave_count	maximum number of slaves to search for at a time
slave_count	current number of slaves known
attempts	number of searches ran

slave_ttl	number of searches before a slave is timed out
initialized	prevent init/removal race conditions
id	w1 bus number
search_count	number of automatic searches to run, -1 unlimited
search_id	allows continuing a search
refcnt	reference count
priv	private data storage
enable_pullup	allows a strong pullup
pullup_duration	time for the next strong pullup
flags	one of w1_master_flags
thread	thread for bus search and netlink commands
mutex	protect most of w1_master
bus_mutex	pretect concurrent bus access
driver	sysfs driver
dev	sysfs device
bus_master	io operations available
seq	sequence number used for netlink broadcasts

Name

struct w1_async_cmd — execute callback from the w1_process kthread

Synopsis

```
struct w1_async_cmd {
  struct list_head async_entry;
  void (* cb) (struct w1_master *dev, struct w1_async_cmd *async_cmd);
};
```

Members

async_entry — link entry

cb — callback function, must list_del and destroy this list before returning

Description

When inserted into the w1_master async_list, w1_process will execute the callback. Embed this into the structure with the command details.

drivers/w1/w1.c

W1 core functions.

Name

w1_search — Performs a ROM Search & registers any devices found.

Synopsis

```
void w1_search (struct w1_master * dev, u8 search_type,
w1_slave_found_callback cb);
```

Arguments

dev	The master device to search
search_type	W1_SEARCH to search all devices, or W1_ALARM_SEARCH to return only devices in the alarmed state
cb	Function to call when a device is found

Description

The 1-wire search is a simple binary tree search. For each bit of the address, we read two bits and write one bit. The bit written will put to sleep all devies that don't match that bit. When the two reads differ, the direction choice is obvious. When both bits are 0, we must choose a path to take. When we can scan all 64 bits without having to choose a path, we are done.

See “Application note 187 1-wire search algorithm” at www.maxim-ic.com

Name

w1_process_callbacks — execute each dev->async_list callback entry

Synopsis

```
int w1_process_callbacks (struct w1_master * dev);
```

Arguments

dev w1_master device

Description

The w1 master list_mutex must be held.

Return

1 if there were commands to executed 0 otherwise

drivers/w1/w1_family.h

Allows registering device family operations.

Name

struct w1_family_ops — operations for a family type

Synopsis

```
struct w1_family_ops {
  int (* add_slave) (struct w1_slave *);
  void (* remove_slave) (struct w1_slave *);
  const struct attribute_group ** groups;
};
```

Members

add_slave	add_slave
remove_slave	remove_slave
groups	sysfs group

Name

struct w1_family — reference counted family structure.

Synopsis

```
struct w1_family {
  struct list_head family_entry;
  u8 fid;
  struct w1_family_ops * fops;
  atomic_t refcnt;
};
```

Members

family_entry	family linked list
fid	8 bit family identifier
fops	operations for this family
refcnt	reference counter

drivers/w1/w1_family.c

Allows registering device family operations.

Name

w1_register_family — register a device family driver

Synopsis

```
int w1_register_family (struct w1_family * newf);
```

Arguments

newf family to register

Name

w1_unregister_family — unregister a device family driver

Synopsis

```
void w1_unregister_family (struct w1_family * fent);
```

Arguments

fent family to unregister

drivers/w1/w1_int.c

W1 internal initialization for master devices.

Name

w1_add_master_device — registers a new master device

Synopsis

```
int w1_add_master_device (struct w1_bus_master * master);
```

Arguments

master master bus device to register

Name

w1_remove_master_device — unregister a master device

Synopsis

```
void w1_remove_master_device (struct w1_bus_master * bm);
```

Arguments

bm master bus device to remove

drivers/w1/w1_netlink.h

W1 external netlink API structures and commands.

Name

enum w1_cn_msg_flags — bitfield flags for struct cn_msg.flags

Synopsis

```
enum w1_cn_msg_flags {
  W1_CN_BUNDLE
};
```

Constants

W1_CN_BUNDLE Request bundling replies into fewer messagse. Be prepared to handle multiple struct cn_msg, struct w1_netlink_msg, and struct w1_netlink_cmd in one packet.

Name

enum w1_netlink_message_types — message type

Synopsis

```
enum w1_netlink_message_types {
  W1_SLAVE_ADD,
  W1_SLAVE_REMOVE,
  W1_MASTER_ADD,
  W1_MASTER_REMOVE,
  W1_MASTER_CMD,
  W1_SLAVE_CMD,
  W1_LIST_MASTERS
};
```

Constants

W1_SLAVE_ADD	notification that a slave device was added
W1_SLAVE_REMOVE	notification that a slave device was removed
W1_MASTER_ADD	notification that a new bus master was added
W1_MASTER_REMOVE	notification that a bus masterwas removed
W1_MASTER_CMD	initiate operations on a specific master
W1_SLAVE_CMD	sends reset, selects the slave, then does a read/write/touch operation
W1_LIST_MASTERS	used to determine the bus master identifiers

Name

struct w1_netlink_msg — holds w1 message type, id, and result

Synopsis

```
struct w1_netlink_msg {
  __u8 type;
  __u8 status;
  __u16 len;
  union id;
  __u8 data[0];
};
```

Members

type	one of enum w1_netlink_message_types
status	kernel feedback for success 0 or errno failure value
len	length of data following w1_netlink_msg
id	union holding master bus id (msg.id) and slave device id (id[8]).
data[0]	start address of any following data

Description

The base message structure for w1 messages over netlink. The netlink connector data sequence is, struct nlmsghdr, struct cn_msg, then one or more struct w1_netlink_msg (each with optional data).

Name

enum w1_commands — commands available for master or slave operations

Synopsis

```
enum w1_commands {
  W1_CMD_READ,
  W1_CMD_WRITE,
  W1_CMD_SEARCH,
  W1_CMD_ALARM_SEARCH,
  W1_CMD_TOUCH,
  W1_CMD_RESET,
  W1_CMD_SLAVE_ADD,
  W1_CMD_SLAVE_REMOVE,
  W1_CMD_LIST_SLAVES,
  W1_CMD_MAX
};
```

Constants

W1_CMD_READ	read len bytes
W1_CMD_WRITE	write len bytes
W1_CMD_SEARCH	initiate a standard search, returns only the slave devices found during that search
W1_CMD_ALARM_SEARCH	search for devices that are currently alarming
W1_CMD_TOUCH	Touches a series of bytes.
W1_CMD_RESET	sends a bus reset on the given master
W1_CMD_SLAVE_ADD	adds a slave to the given master, 8 byte slave id at data[0]
W1_CMD_SLAVE_REMOVE	removes a slave to the given master, 8 byte slave id at data[0]
W1_CMD_LIST_SLAVES	list of slaves registered on this master
W1_CMD_MAX	number of available commands

Name

struct w1_netlink_cmd — holds the command and data

Synopsis

```
struct w1_netlink_cmd {
  __u8 cmd;
  __u8 res;
  __u16 len;
  __u8 data[0];
};
```

Members

cmd	one of enum w1_commands
res	reserved
len	length of data following w1_netlink_cmd
data[0]	start address of any following data

Description

One or more struct w1_netlink_cmd is placed starting at w1_netlink_msg.data each with optional data.

drivers/w1/w1_io.c

W1 input/output.

Name

w1_write_8 — Writes 8 bits.

Synopsis

```
void w1_write_8 (struct w1_master * dev, u8 byte);
```

Arguments

dev the master device

byte the byte to write

Name

w1_read_8 — Reads 8 bits.

Synopsis

```
u8 w1_read_8 (struct w1_master * dev);
```

Arguments

dev the master device

Return

the byte read

Name

w1_write_block — Writes a series of bytes.

Synopsis

```
void w1_write_block (struct w1_master * dev, const u8 * buf, int len);
```

Arguments

dev the master device

buf pointer to the data to write

len the number of bytes to write

Name

w1_touch_block — Touches a series of bytes.

Synopsis

```
void w1_touch_block (struct w1_master * dev, u8 * buf, int len);
```

Arguments

dev the master device

buf pointer to the data to write

len the number of bytes to write

Name

w1_read_block — Reads a series of bytes.

Synopsis

```
u8 w1_read_block (struct w1_master * dev, u8 * buf, int len);
```

Arguments

dev the master device

buf pointer to the buffer to fill

len the number of bytes to read

Return

the number of bytes read

Name

w1_reset_bus — Issues a reset bus sequence.

Synopsis

```
int w1_reset_bus (struct w1_master * dev);
```

Arguments

dev the master device

Return

0=Device present, 1=No device present or error

Name

w1_reset_select_slave — reset and select a slave

Synopsis

```
int w1_reset_select_slave (struct w1_slave * sl);
```

Arguments

sl the slave to select

Description

Resets the bus and then selects the slave by sending either a skip rom or a rom match. A skip rom is issued if there is only one device registered on the bus. The w1 master lock must be held.

Return

0=success, anything else=error

Name

w1_reset_resume_command — resume instead of another match ROM

Synopsis

```
int w1_reset_resume_command (struct w1_master * dev);
```

Arguments

dev the master device

Description

When the workflow with a slave amongst many requires several successive commands a reset between each, this function is similar to doing a reset then a match ROM for the last matched ROM. The advantage being that the matched ROM step is skipped in favor of the resume command. The slave must support the command of course.

If the bus has only one slave, traditionnaly the match ROM is skipped and a “SKIP ROM” is done for efficiency. On multi-slave busses, this doesn't work of course, but the resume command is the next best thing.

The w1 master lock must be held.

Name

w1_next_pullup — register for a strong pullup

Synopsis

```
void w1_next_pullup (struct w1_master * dev, int delay);
```

Arguments

dev the master device

delay time in milliseconds

Description

Put out a strong pull-up of the specified duration after the next write operation. Not all hardware supports strong pullups. Hardware that doesn't support strong pullups will sleep for the given time after the write operation without a strong pullup. This is a one shot request for the next write, specifying zero will clear a previous request. The w1 master lock must be held.

Return

0=success, anything else=error

Name

w1_touch_bit — Generates a write-0 or write-1 cycle and samples the level.

Synopsis

```
u8 w1_touch_bit (struct w1_master * dev, int bit);
```

Arguments

dev the master device

bit 0 - write a 0, 1 - write a 0 read the level

Name

w1_write_bit — Generates a write-0 or write-1 cycle.

Synopsis

```
void w1_write_bit (struct w1_master * dev, int bit);
```

Arguments

dev the master device

bit bit to write

Description

Only call if dev->bus_master->touch_bit is NULL

Name

w1_pre_write — pre-write operations

Synopsis

```
void w1_pre_write (struct w1_master * dev);
```

Arguments

dev the master device

Description

Pre-write operation, currently only supporting strong pullups. Program the hardware for a strong pullup, if one has been requested and the hardware supports it.

Name

w1_post_write — post-write options

Synopsis

```
void w1_post_write (struct w1_master * dev);
```

Arguments

dev the master device

Description

Post-write operation, currently only supporting strong pullups. If a strong pullup was requested, clear it if the hardware supports them, or execute the delay otherwise, in either case clear the request.

Name

w1_read_bit — Generates a write-1 cycle and samples the level.

Synopsis

```
u8 w1_read_bit (struct w1_master * dev);
```

Arguments

dev the master device

Description

Only call if dev->bus_master->touch_bit is NULL

Name

w1_triplet — * Does a triplet - used for searching ROM addresses.

Synopsis

```
u8 w1_triplet (struct w1_master * dev, int bdir);
```

Arguments

dev the master device

bdir the bit to write if both id_bit and comp_bit are 0

Return bits

bit 0 = id_bit bit 1 = comp_bit bit 2 = dir_taken If both bits 0 & 1 are set, the search should be restarted.

Return

bit fields - see above