Linux W1 Documentation

The kernel development community

Jul 14, 2020

CONTENTS

CHAPTER

ONE

INTRODUCTION TO THE 1-WIRE (W1) SUBSYSTEM

The 1-wire bus is a simple master-slave bus that communicates via a single signal wire (plus ground, so two wires).

Devices communicate on the bus by pulling the signal to ground via an open drain output and by sampling the logic level of the signal line.

The w1 subsystem provides the framework for managing w1 masters and communication with slaves.

All w1 slave devices must be connected to a w1 bus master device.

Example w1 master devices:

- DS9490 usb device
- W1-over-GPIO
- DS2482 (i2c to w1 bridge)
- Emulated devices, such as a RS232 converter, parallel port adapter, etc

1.1 What does the w1 subsystem do?

When a w1 master driver registers with the w1 subsystem, the following occurs:

- sysfs entries for that w1 master are created
- the w1 bus is periodically searched for new slave devices

When a device is found on the bus, w1 core tries to load the driver for its family and check if it is loaded. If so, the family driver is attached to the slave. If there is no driver for the family, default one is assigned, which allows to perform almost any kind of operations. Each logical operation is a transaction in nature, which can contain several (two or one) low-level operations. Let’s see how one can read EEPROM context: 1. one must write control buffer, i.e. buffer containing command byte and two byte address. At this step bus is reset and appropriate device is selected using either W1_SKIP_ROM or W1_MATCH_ROM command. Then provided control buffer is being written to the wire. 2. reading. This will issue reading eeprom response.

It is possible that between 1. and 2. w1 master thread will reset bus for searching and slave device will be even removed, but in this case 0xff will be read, since no device was selected.

1.2 W1 device families

Slave devices are handled by a driver written for a family of w1 devices.

A family driver populates a struct w1_family_ops (see w1_family.h) and registers with the w1 subsystem.

Current family drivers:

w1_therm

- (ds18?20 thermal sensor family driver) provides temperature reading function which is bound to ->rbin() method of the above w1_family_ops structure.

w1_smem

- driver for simple 64bit memory cell provides ID reading method.

You can call above methods by reading appropriate sysfs files.

1.3 What does a w1 master driver need to implement?

The driver for w1 bus master must provide at minimum two functions.

Emulated devices must provide the ability to set the output signal level (write_bit) and sample the signal level (read_bit).

Devices that support the 1-wire natively must provide the ability to write and sample a bit (touch_bit) and reset the bus (reset_bus).

Most hardware provides higher-level functions that offload w1 handling. See struct w1_bus_master definition in w1.h for details.

1.4 w1 master sysfs interface

<xx-xxxxxxxxxxxx>	A directory for a found device. The format is family-serial
bus	(standard) symlink to the w1 bus
driver	(standard) symlink to the w1 driver
w1_master_add	(rw) manually register a slave device
w1_master_attempts	(ro) the number of times a search was attempted
w1_master_max_slave_count	(rw) maximum number of slaves to search for at a time
w1_master_name	(ro) the name of the device (w1_bus_masterX)
w1_master_pullup	(rw) 5V strong pullup 0 enabled, 1 disabled
w1_master_remove	(rw) manually remove a slave device
w1_master_search	(rw) the number of searches left to do, -1=continual (default)
w1_master_slave_count	(ro) the number of slaves found
w1_master_slaves	(ro) the names of the slaves, one per line
w1_master_timeout	(ro) the delay in seconds between searches
w1_master_timeout_us	(ro) the delay in microseconds beetwen searches

If you have a w1 bus that never changes (you don't add or remove devices), you can set the module parameter search_count to a small positive number for an initially small number of bus searches. Alternatively it could be set to zero, then manually add the slave device serial numbers by w1_master_add device file. The w1_master_add and w1_master_remove files generally only make sense when searching is disabled, as a search will redetect manually removed devices that are present and timeout manually added devices that aren't on the bus.

Bus searches occur at an interval, specified as a summ of timeout and timeout_us module parameters (either of which may be 0) for as long as w1_master_search remains greater than 0 or is -1. Each search attempt decrements w1_master_search by 1 (down to 0) and increments w1_master_attempts by 1.

1.5 w1 slave sysfs interface

bus	(standard) symlink to the w1 bus
driver	(standard) symlink to the w1 driver
name	the device name, usually the same as the directory name
w1_slave	(optional) a binary file whose meaning depends on the family driver
rw	(optional) created for slave devices which do not have appropriate family driver. Allows to read/write binary data.

CHAPTER

TWO

USERSPACE COMMUNICATION PROTOCOL OVER CONNECTOR

2.1 Message types

There are three types of messages between w1 core and userspace:

1. Events. They are generated each time a new master or slave device is found either due to automatic or requested search.
2. Userspace commands.
3. Replies to userspace commands.

2.2 Protocol

```
[struct cn_msg] - connector header.
      Its length field is equal to size of the attached data
[struct w1_netlink_msg] - w1 netlink header.
      __u8 type       - message type.
                      W1_LIST_MASTERS
                              list current bus masters
                      W1_SLAVE_ADD/W1_SLAVE_REMOVE
                              slave add/remove events
                      W1_MASTER_ADD/W1_MASTER_REMOVE
                              master add/remove events
                      W1_MASTER_CMD
                              userspace command for bus master
                              device (search/alarm search)
                      W1_SLAVE_CMD
                              userspace command for slave device
                              (read/write/touch)
      __u8 status     - error indication from kernel
      __u16 len       - size of data attached to this header data
      union {
              __u8 id[8];                     - slave unique device id
              struct w1_mst {
                      __u32           id;     - master's id
                      __u32           res;    - reserved
              } mst;
      } id;
```

(continues on next page)

(continued from previous page)

```
[struct w1_netlink_cmd] - command for given master or slave device.
      __u8 cmd        - command opcode.
                      W1_CMD_READ     - read command
                      W1_CMD_WRITE    - write command
                      W1_CMD_SEARCH   - search command
                      W1_CMD_ALARM_SEARCH - alarm search command
                      W1_CMD_TOUCH    - touch command
                              (write and sample data back to userspace)
                      W1_CMD_RESET    - send bus reset
                      W1_CMD_SLAVE_ADD        - add slave to kernel list
                      W1_CMD_SLAVE_REMOVE     - remove slave from kernel␣
↪list
                      W1_CMD_LIST_SLAVES      - get slaves list from kernel
      __u8 res        - reserved
      __u16 len       - length of data for this command
              For read command data must be allocated like for write␣
↪command
      __u8 data[0]    - data for this command
```

Each connector message can include one or more w1_netlink_msg with zero or more attached w1_netlink_cmd messages.

For event messages there are no w1_netlink_cmd embedded structures, only connector header and w1_netlink_msg strucutre with "len" field being zero and filled type (one of event types) and id: either 8 bytes of slave unique id in host order, or master's id, which is assigned to bus master device when it is added to w1 core.

Currently replies to userspace commands are only generated for read command request. One reply is generated exactly for one w1_netlink_cmd read request. Replies are not combined when sent - i.e. typical reply messages looks like the following:

```
[cn_msg][w1_netlink_msg][w1_netlink_cmd]
cn_msg.len = sizeof(struct w1_netlink_msg) +
           sizeof(struct w1_netlink_cmd) +
           cmd->len;
w1_netlink_msg.len = sizeof(struct w1_netlink_cmd) + cmd->len;
w1_netlink_cmd.len = cmd->len;
```

Replies to W1_LIST_MASTERS should send a message back to the userspace which will contain list of all registered master ids in the following format:

```
cn_msg (CN_W1_IDX.CN_W1_VAL as id, len is equal to sizeof(struct
w1_netlink_msg) plus number of masters multiplied by 4)
w1_netlink_msg (type: W1_LIST_MASTERS, len is equal to
        number of masters multiplied by 4 (u32 size))
id0 ... idN
```

Each message is at most 4k in size, so if number of master devices exceeds this, it will be split into several messages.

W1 search and alarm search commands.

request:

```
[cn_msg]
  [w1_netlink_msg type = W1_MASTER_CMD
      id is equal to the bus master id to use for searching]
  [w1_netlink_cmd cmd = W1_CMD_SEARCH or W1_CMD_ALARM_SEARCH]
```

reply:

```
[cn_msg, ack = 1 and increasing, 0 means the last message,
      seq is equal to the request seq]
[w1_netlink_msg type = W1_MASTER_CMD]
[w1_netlink_cmd cmd = W1_CMD_SEARCH or W1_CMD_ALARM_SEARCH
      len is equal to number of IDs multiplied by 8]
[64bit-id0 ... 64bit-idN]
```

Length in each header corresponds to the size of the data behind it, so w1_netlink_cmd->len = N * 8; where N is number of IDs in this message. Can be zero.

```
w1_netlink_msg->len = sizeof(struct w1_netlink_cmd) + N * 8;
cn_msg->len = sizeof(struct w1_netlink_msg) +
            sizeof(struct w1_netlink_cmd) +
            N*8;
```

W1 reset command:

```
[cn_msg]
  [w1_netlink_msg type = W1_MASTER_CMD
      id is equal to the bus master id to use for searching]
  [w1_netlink_cmd cmd = W1_CMD_RESET]
```

2.3 Command status replies

Each command (either root, master or slave with or without w1_netlink_cmd structure) will be 'acked' by the w1 core. Format of the reply is the same as request message except that length parameters do not account for data requested by the user, i.e. read/write/touch IO requests will not contain data, so w1_netlink_cmd.len will be 0, w1_netlink_msg.len will be size of the w1_netlink_cmd structure and cn_msg.len will be equal to the sum of the sizeof(struct w1_netlink_msg) and sizeof(struct w1_netlink_cmd). If reply is generated for master or root command (which do not have w1_netlink_cmd attached), reply will contain only cn_msg and w1_netlink_msg structures.

w1_netlink_msg.status field will carry positive error value (EINVAL for example) or zero in case of success.

All other fields in every structure will mirror the same parameters in the request message (except lengths as described above).

Status reply is generated for every w1_netlink_cmd embedded in the w1_netlink_msg, if there are no w1_netlink_cmd structures, reply will be generated for the w1_netlink_msg.

All w1_netlink_cmd command structures are handled in every w1_netlink_msg, even if there were errors, only length mismatch interrupts message processing.

2.4 Operation steps in w1 core when new command is received

When new message (w1_netlink_msg) is received w1 core detects if it is master or slave request, according to w1_netlink_msg.type field. Then master or slave device is searched for. When found, master device (requested or those one on where slave device is found) is locked. If slave command is requested, then reset/select procedure is started to select given device.

Then all requested in w1_netlink_msg operations are performed one by one. If command requires reply (like read command) it is sent on command completion.

When all commands (w1_netlink_cmd) are processed master device is unlocked and next w1_netlink_msg header processing started.

2.5 Connector [1] specific documentation

Each connector message includes two u32 fields as “address”. w1 uses CN_W1_IDX and CN_W1_VAL defined in include/linux/connector.h header. Each message also includes sequence and acknowledge numbers. Sequence number for event messages is appropriate bus master sequence number increased with each event message sent “through” this master. Sequence number for userspace requests is set by userspace application. Sequence number for reply is the same as was in request, and acknowledge number is set to seq+1.

2.6 Additional documentation, source code examples

1. Documentation/driver-api/connector.rst
2. http://www.ioremap.net/archive/w1

   This archive includes userspace application w1d.c which uses read/write/search commands for all master/slave devices found on the bus.

CHAPTER

THREE

1-WIRE MASTER DRIVERS

3.1 Kernel driver ds2482

Supported chips:

- Maxim DS2482-100, Maxim DS2482-800

  Prefix: 'ds2482'

  Addresses scanned: None

  Datasheets:

  - http://datasheets.maxim-ic.com/en/ds/DS2482-100.pdf
  - http://datasheets.maxim-ic.com/en/ds/DS2482-800.pdf

Author: Ben Gardner <bgardner@wabtec.com>

3.1.1 Description

The Maxim/Dallas Semiconductor DS2482 is a I2C device that provides one (DS2482-100) or eight (DS2482-800) 1-wire busses.

3.1.2 General Remarks

Valid addresses are 0x18, 0x19, 0x1a, and 0x1b.

However, the device cannot be detected without writing to the i2c bus, so no detection is done. You should instantiate the device explicitly.

```
$ modprobe ds2482
$ echo ds2482 0x18 > /sys/bus/i2c/devices/i2c-0/new_device
```

3.2 Kernel driver ds2490

Supported chips:

- Maxim DS2490 based

Author: Evgeniy Polyakov <johnpol@2ka.mipt.ru>

3.2.1 Description

The Maxim/Dallas Semiconductor DS2490 is a chip which allows to build USB <-> W1 bridges.

DS9490(R) is a USB <-> W1 bus master device which has 0x81 family ID integrated chip and DS2490 low-level operational chip.

Notes and limitations.

- The weak pullup current is a minimum of 0.9mA and maximum of 6.0mA.
- The 5V strong pullup is supported with a minimum of 5.9mA and a maximum of 30.4 mA. (From DS2490.pdf)
- The hardware will detect when devices are attached to the bus on the next bus (reset?) operation, however only a message is printed as the core w1 code doesn't make use of the information. Connecting one device tends to give multiple new device notifications.
- The number of USB bus transactions could be reduced if w1_reset_send was added to the API. The name is just a suggestion. It would take a write buffer and a read buffer (along with sizes) as arguments. The ds2490 block I/O command supports reset, write buffer, read buffer, and strong pullup all in one command, instead of the current 1 reset bus, 2 write the match rom command and slave rom id, 3 block write and read data. The write buffer needs to have the match rom command and slave rom id prepended to the front of the requested write buffer, both of which are known to the driver.
- The hardware supports normal, flexible, and overdrive bus communication speeds, but only the normal is supported.
- The registered w1_bus_master functions don't define error conditions. If a bus search is in progress and the ds2490 is removed it can produce a good amount of error output before the bus search finishes.
- The hardware supports detecting some error conditions, such as short, alarming presence on reset, and no presence on reset, but the driver doesn't query those values.
- The ds2490 specification doesn't cover short bulk in reads in detail, but my observation is if fewer bytes are requested than are available, the bulk read will return an error and the hardware will clear the entire bulk in buffer. It would be possible to read the maximum buffer size to not run into this error condition, only extra bytes in the buffer is a logic error in the driver. The code should should match reads and writes as well as data sizes. Reads and writes are serialized and the status verifies that the chip is idle (and data is available) before the read is executed, so it should not happen.

- Running x86_64 2.6.24 UHCI under qemu 0.9.0 under x86_64 2.6.22-rc6 with a OHCI controller, ds2490 running in the guest would operate normally the first time the module was loaded after qemu attached the ds2490 hardware, but if the module was unloaded, then reloaded most of the time one of the bulk out or in, and usually the bulk in would fail. qemu sets a 50ms timeout and the bulk in would timeout even when the status shows data available. A bulk out write would show a successful completion, but the ds2490 status register would show 0 bytes written. Detaching qemu from the ds2490 hardware and reattaching would clear the problem. usbmon output in the guest and host did not explain the problem. My guess is a bug in either qemu or the host OS and more likely the host OS.

03-06-2008 David Fries <David@Fries.net>

3.3 Kernel driver mxc_w1

Supported chips:

- Freescale MX27, MX31 and probably other i.MX SoCs

  Datasheets:

  - http://www.freescale.com/files/32bit/doc/data_sheet/MCIMX31.pdf?fpsp=1
  - http://cache.freescale.com/files/dsp/doc/archive/MCIMX27.pdf?fsrch=1&WT_TYPE=Data%20Sheets&WT_VENDOR=FREESCALE&WT_FILE_FORMAT=pdf&WT_ASSET=Documentation

Author:

Originally based on Freescale code, prepared for mainline by Sascha Hauer <s.hauer@pengutronix.de>

3.4 Kernel driver for omap HDQ/1-wire module

3.4.1 Supported chips:

HDQ/1-wire controller on the TI OMAP 2430/3430 platforms.

3.4.2 A useful link about HDQ basics:

http://focus.ti.com/lit/an/slua408a/slua408a.pdf

3.4.3 Description:

The HDQ/1-Wire module of TI OMAP2430/3430 platforms implement the hardware protocol of the master functions of the Benchmark HDQ and the Dallas Semiconductor 1-Wire protocols. These protocols use a single wire for communication between the master (HDQ/1-Wire controller) and the slave (HDQ/1-Wire external compliant device).

A typical application of the HDQ/1-Wire module is the communication with battery monitor (gas gauge) integrated circuits.

The controller supports operation in both HDQ and 1-wire mode. The essential difference between the HDQ and 1-wire mode is how the slave device responds to initialization pulse.In HDQ mode, the firmware does not require the host to create an initialization pulse to the slave.However, the slave can be reset by using an initialization pulse (also referred to as a break pulse).The slave does not respond with a presence pulse as it does in the 1-Wire protocol.

3.4.4 Remarks:

The driver (drivers/w1/masters/omap_hdq.c) supports the HDQ mode of the controller. In this mode, as we can not read the ID which obeys the W1 spec(family:id:crc), a module parameter can be passed to the driver which will be used to calculate the CRC and pass back an appropriate slave ID to the W1 core.

By default the master driver and the BQ slave i/f driver(drivers/w1/slaves/w1_bq27000.c) sets the ID to 1. Please note to load both the modules with a different ID if required, but note that the ID used should be same for both master and slave driver loading.

e.g:

```
insmod omap_hdq.ko W1_ID=2
insmod w1_bq27000.ko F_ID=2
```

The driver also supports 1-wire mode. In this mode, there is no need to pass slave ID as parameter. The driver will auto-detect slaves connected to the bus using SEARCH_ROM procedure. 1-wire mode can be selected by setting "ti,mode" property to "1w" in DT (see Documentation/devicetree/bindings/w1/omap-hdq.txt for more details). By default driver is in HDQ mode.

3.5 Kernel driver w1-gpio

Author: Ville Syrjala <syrjala@sci.fi>

3.5.1 Description

GPIO 1-wire bus master driver. The driver uses the GPIO API to control the wire and the GPIO pin can be specified using GPIO machine descriptor tables. It is also possible to define the master using device tree, see Documentation/devicetree/bindings/w1/w1-gpio.txt

3.5.2 Example (mach-at91)

```
#include <linux/gpio/machine.h>
#include <linux/w1-gpio.h>

static struct gpiod_lookup_table foo_w1_gpiod_table = {
      .dev_id = "w1-gpio",
      .table = {
              GPIO_LOOKUP_IDX("at91-gpio", AT91_PIN_PB20, NULL, 0,
                      GPIO_ACTIVE_HIGH|GPIO_OPEN_DRAIN),
      },
};

static struct w1_gpio_platform_data foo_w1_gpio_pdata = {
      .ext_pullup_enable_pin  = -EINVAL,
};

static struct platform_device foo_w1_device = {
      .name                   = "w1-gpio",
      .id                     = -1,
      .dev.platform_data      = &foo_w1_gpio_pdata,
};

...
      at91_set_GPIO_periph(foo_w1_gpio_pdata.pin, 1);
      at91_set_multi_drive(foo_w1_gpio_pdata.pin, 1);
      gpiod_add_lookup_table(&foo_w1_gpiod_table);
      platform_device_register(&foo_w1_device);
```

. SPDX-License-Identifier: GPL-2.0

CHAPTER

FOUR

1-WIRE SLAVE DRIVERS

4.1 w1_ds2406 kernel driver

Supported chips:

- Maxim DS2406 (and other family 0x12) addressable switches

Author: Scott Alfter <scott@alfter.us>

4.1.1 Description

The w1_ds2406 driver allows connected devices to be switched on and off. These chips also provide 128 bytes of OTP EPROM, but reading/writing it is not supported. In TSOC-6 form, the DS2406 provides two switch outputs and can be provided with power on a dedicated input. In TO-92 form, it provides one output and uses parasitic power only.

The driver provides two sysfs files. state is readable; it gives the current state of each switch, with PIO A in bit 0 and PIO B in bit 1. The driver ORs this state with 0x30, so shell scripts get an ASCII 0/1/2/3 to work with. output is writable; bits 0 and 1 control PIO A and B, respectively. Bits 2-7 are ignored, so it' s safe to write ASCII data.

CRCs are checked on read and write. Failed checks cause an I/O error to be returned. On a failed write, the switch status is not changed.

4.2 Kernel driver w1_ds2413

Supported chips:

- Maxim DS2413 1-Wire Dual Channel Addressable Switch

supported family codes:

W1_FAMILY_DS2413	0x3A

Author: Mariusz Bialonczyk <manio@skyboo.net>

4.2.1 Description

The DS2413 chip has two open-drain outputs (PIO A and PIO B). Support is provided through the sysfs files “output” and “state”.

4.2.2 Reading state

The “state” file provides one-byte value which is in the same format as for the chip PIO_ACCESS_READ command (refer the datasheet for details):

Bit 0:	PIOA Pin State
Bit 1:	PIOA Output Latch State
Bit 2:	PIOB Pin State
Bit 3:	PIOB Output Latch State
Bit 4-7:	Complement of Bit 3 to Bit 0 (verified by the kernel module)

This file is readonly.

4.2.3 Writing output

You can set the PIO pins using the “output” file. It is writable, you can write one-byte value to this sysfs file. Similarly the byte format is the same as for the PIO_ACCESS_WRITE command:

Bit 0:	PIOA
Bit 1:	PIOB
Bit 2-7:	No matter (driver will set it to “1” s)

The chip has some kind of basic protection against transmission errors. When reading the state, there is a four complement bits. The driver is checking this complement, and when it is wrong then it is returning I/O error.

When writing output, the master must repeat the PIO Output Data byte in its inverted form and it is waiting for a confirmation. If the write is unsuccessful for three times, the write also returns I/O error.

4.3 Kernel driver w1_ds2423

Supported chips:

- Maxim DS2423 based counter devices.

supported family codes:

W1_THERM_DS2423	0x1D

Author: Mika Laitio <lamikr@pilppa.org>

4.3.1 Description

Support is provided through the sysfs w1_slave file. Each opening and read sequence of w1_slave file initiates the read of counters and ram available in DS2423 pages 12 - 15.

Result of each page is provided as an ASCII output where each counter value and associated ram buffer is outpputed to own line.

Each lines will contain the values of 42 bytes read from the counter and memory page along the crc=YES or NO for indicating whether the read operation was successful and CRC matched. If the operation was successful, there is also in the end of each line a counter value expressed as an integer after c=

Meaning of 42 bytes represented is following:

- 1 byte from ram page
- 4 bytes for the counter value
- 4 zero bytes
- 2 bytes for crc16 which was calculated from the data read since the previous crc bytes
- 31 remaining bytes from the ram page
- crc=YES/NO indicating whether read was ok and crc matched
- c=<int> current counter value

example from the successful read:

```
00 02 00 00 00 00 00 00 00 6d 38 00 ff ff 00 00 fe ff 00 00 ff ff 00 00 ff␣
↪ff 00 00 ff ff 00 00 ff ff 00 00 ff ff 00 00 ff ff crc=YES c=2
00 02 00 00 00 00 00 00 00 e0 1f 00 ff ff 00 00 ff ff 00 00 ff ff 00 00 ff␣
↪ff 00 00 ff ff 00 00 ff ff 00 00 ff ff 00 00 ff ff crc=YES c=2
00 29 c6 5d 18 00 00 00 00 04 37 00 ff ff 00 00 ff ff 00 00 ff ff 00 00 ff␣
↪ff 00 00 ff ff 00 00 ff ff 00 00 ff ff 00 00 ff ff crc=YES c=408798761
00 05 00 00 00 00 00 00 00 8d 39 ff ff ff ff ff ff ff ff ff ff ff ff ff ff␣
↪ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff crc=YES c=5
```

example from the read with crc errors:

```
00 02 00 00 00 00 00 00 00 6d 38 00 ff ff 00 00 fe ff 00 00 ff ff 00 00 ff␣
↪ff 00 00 ff ff 00 00 ff ff 00 00 ff ff 00 00 ff ff crc=YES c=2
00 02 00 00 22 00 00 00 00 e0 1f 00 ff ff 00 00 ff ff 00 00 ff ff 00 00 ff␣
↪ff 00 00 ff ff 00 00 ff ff 00 00 ff ff 00 00 ff ff crc=NO
00 e1 61 5d 19 00 00 00 00 df 0b 00 ff ff 00 00 ff ff 00 00 ff ff 00 00 ff␣
↪ff 00 00 ff ff 00 00 ff ff 00 00 ff ff 00 00 ff ff crc=NO
00 05 00 00 20 00 00 00 00 8d 39 ff ff ff ff ff ff ff ff ff ff ff ff ff ff␣
↪ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff crc=NO
```

4.4 Kernel driver w1_ds2438

Supported chips:

- Maxim DS2438 Smart Battery Monitor

supported family codes:

W1_FAMILY_DS2438	0x26

Author: Mariusz Bialonczyk <manio@skyboo.net>

4.4.1 Description

The DS2438 chip provides several functions that are desirable to carry in a battery pack. It also has a 40 bytes of nonvolatile EEPROM. Because the ability of temperature, current and voltage measurement, the chip is also often used in weather stations and applications such as: rain gauge, wind speed/direction measuring, humidity sensing, etc.

Current support is provided through the following sysfs files (all files except "iad" are readonly):

4.4.2 "iad"

This file controls the 'Current A/D Control Bit' (IAD) in the Status/Configuration Register. Writing a zero value will clear the IAD bit and disables the current measurements. Writing value "1" is setting the IAD bit (enables the measurements). The IAD bit is enabled by default in the DS2438.

When writing to sysfs file bits 2-7 are ignored, so it's safe to write ASCII. An I/O error is returned when there is a problem setting the new value.

4.4.3 "page0"

This file provides full 8 bytes of the chip Page 0 (00h). This page contains the most frequently accessed information of the DS2438. Internally when this file is read, the additional CRC byte is also obtained from the slave device. If it is correct, the 8 bytes page data are passed to userspace, otherwise an I/O error is returned.

4.4.4 "temperature"

Opening and reading this file initiates the CONVERT_T (temperature conversion) command of the chip, afterwards the temperature is read from the device registers and provided as an ASCII decimal value.

Important: The returned value has to be divided by 256 to get a real temperature in degrees Celsius.

4.4.5 “vad” , “vdd”

Opening and reading this file initiates the CONVERT_V (voltage conversion) command of the chip.

Depending on a sysfs filename a different input for the A/D will be selected:

vad: general purpose A/D input (VAD)

vdd: battery input (VDD)

After the voltage conversion the value is returned as decimal ASCII. Note: To get a volts the value has to be divided by 100.

4.5 Kernel driver w1_ds28e04

Supported chips:

- Maxim DS28E04-100 4096-Bit Addressable 1-Wire EEPROM with PIO

supported family codes:

W1_FAMILY_DS28E04	0x1C

Author: Markus Franke, <franke.m@sebakmt.com> <franm@hrz.tu-chemnitz.de>

4.5.1 Description

Support is provided through the sysfs files “eeprom” and “pio” . CRC checking during memory accesses can optionally be enabled/disabled via the device attribute “crccheck” . The strong pull-up can optionally be enabled/disabled via the module parameter “w1_strong_pullup” .

Memory Access

> A read operation on the “eeprom” file reads the given amount of bytes from the EEPROM of the DS28E04.
>
> A write operation on the “eeprom” file writes the given byte sequence to the EEPROM of the DS28E04. If CRC checking mode is enabled only fully aligned blocks of 32 bytes with valid CRC16 values (in bytes 30 and 31) are allowed to be written.

PIO Access

> The 2 PIOs of the DS28E04-100 are accessible via the “pio” sysfs file.
>
> The current status of the PIO’ s is returned as an 8 bit value. Bit 0/1 represent the state of PIO_0/PIO_1. Bits 2..7 do not care. The PIO’ s are driven low-active, i.e. the driver delivers/expects low-active values.

4.6 Kernel driver w1_ds28e17

Supported chips:

- Maxim DS28E17 1-Wire-to-I2C Master Bridge

supported family codes:

W1_FAMILY_DS28E17	0x19

Author: Jan Kandziora <jjj@gmx.de>

4.6.1 Description

The DS28E17 is a Onewire slave device which acts as an I2C bus master.

This driver creates a new I2C bus for any DS28E17 device detected. I2C buses come and go as the DS28E17 devices come and go. I2C slave devices connected to a DS28E17 can be accessed by the kernel or userspace tools as if they were connected to a "native" I2C bus master.

An udev rule like the following:

```
SUBSYSTEM=="i2c-dev", KERNEL=="i2c-[0-9]*", ATTRS{name}=="w1-19-*", \
        SYMLINK+="i2c-$attr{name}"
```

may be used to create stable /dev/i2c- entries based on the unique id of the DS28E17 chip.

Driver parameters are:

speed: This sets up the default I2C speed a DS28E17 get configured for as soon it is connected. The power-on default of the DS28E17 is 400kBaud, but chips may come and go on the Onewire bus without being de-powered and as soon the "w1_ds28e17" driver notices a freshly connected, or reconnected DS28E17 device on the Onewire bus, it will re-apply this setting.

Valid values are 100, 400, 900 [kBaud]. Any other value means to leave alone the current DS28E17 setting on detect. The default value is 100.

stretch: This sets up the default stretch value used for freshly connected DS28E17 devices. It is a multiplier used on the calculation of the busy wait time for an I2C transfer. This is to account for I2C slave devices which make heavy use of the I2C clock stretching feature and thus, the needed timeout cannot be pre-calculated correctly. As the w1_ds28e17 driver checks the DS28E17's busy flag in a loop after the precalculated wait time, it should be hardly needed to tweak this setting.

Leave it at 1 unless you get ETIMEDOUT errors and a "w1_slave_driver 19-00000002dbd8: busy timeout" in the kernel log.

Valid values are 1 to 9. The default is 1.

The driver creates sysfs files /sys/bus/w1/devices/19-<id>/speed and /sys/bus/w1/devices/19-<id>/stretch for each device, preloaded with the default settings from the driver parameters. They may be changed anytime. In addition a directory /sys/bus/w1/devices/19-<id>/i2c-<nnn> for the I2C bus master sysfs structure is created.

See https://github.com/ianka/w1_ds28e17 for even more information.

4.7 Kernel driver w1_therm

Supported chips:

- Maxim ds18*20 based temperature sensors.
- Maxim ds1825 based temperature sensors.

Author: Evgeniy Polyakov <johnpol@2ka.mipt.ru>

4.7.1 Description

w1_therm provides basic temperature conversion for ds18*20 devices, and the ds28ea00 device.

Supported family codes:

W1_THERM_DS18S20	0x10
W1_THERM_DS1822	0x22
W1_THERM_DS18B20	0x28
W1_THERM_DS1825	0x3B
W1_THERM_DS28EA00	0x42

Support is provided through the sysfs w1_slave file. Each open and read sequence will initiate a temperature conversion then provide two lines of ASCII output. The first line contains the nine hex bytes read along with a calculated crc value and YES or NO if it matched. If the crc matched the returned values are retained. The second line displays the retained values along with a temperature in millidegrees Centigrade after t=.

Alternatively, temperature can be read using temperature sysfs, it return only temperature in millidegrees Centigrade.

A bulk read of all devices on the bus could be done writing 'trigger' in the therm_bulk_read sysfs entry at w1_bus_master level. This will sent the convert command on all devices on the bus, and if parasite powered devices are detected on the bus (and strong pullup is enable in the module), it will drive the line high during the longer conversion time required by parasited powered device on the line. Reading therm_bulk_read will return 0 if no bulk conversion pending, -1 if at least one sensor still in conversion, 1 if conversion is complete but at least one sensor value has not been read yet. Result temperature is then accessed by reading the temperature sysfs entry of each device, which may return empty if conversion is still in progress. Note that if a bulk read is sent but one sensor is not read immediately, the next access to temperature on this device will return the temperature measured at the time of issue of the bulk read command (not the current temperature).

Writing a value between 9 and 12 to the sysfs w1_slave file will change the precision of the sensor for the next readings. This value is in (volatile) SRAM, so it is reset when the sensor gets power-cycled.

To store the current precision configuration into EEPROM, the value 0 has to be written to the sysfs w1_slave file. Since the EEPROM has a limited amount of writes (>50k), this command should be used wisely.

Alternatively, resolution can be set or read (value from 9 to 12) using the dedicated resolution sysfs entry on each device. This sysfs entry is not present for devices not supporting this feature. Driver will adjust the correct conversion time for each device regarding to its resolution setting. In particular, strong pullup will be applied if required during the conversion duration.

The write-only sysfs entry eeprom is an alternative for EEPROM operations:

- 'save' : will save device RAM to EEPROM
- 'restore' : will restore EEPROM data in device RAM.

ext_power syfs entry allow tho check the power status of each device.

- '0' : device parasite powered
- '1' : device externally powered

sysfs alarms allow read or write TH and TL (Temperature High an Low) alarms. Values shall be space separated and in the device range (typical -55 degC to 125 degC). Values are integer as they are store in a 8bit register in the device. Lowest value is automatically put to TL.Once set, alarms could be search at master level.

The module parameter strong_pullup can be set to 0 to disable the strong pullup, 1 to enable autodetection or 2 to force strong pullup. In case of autodetection, the driver will use the "READ POWER SUPPLY" command to check if there are pariste powered devices on the bus. If so, it will activate the master's strong pullup. In case the detection of parasite devices using this command fails (seems to be the case with some DS18S20) the strong pullup can be force-enabled.

If the strong pullup is enabled, the master's strong pullup will be driven when the conversion is taking place, provided the master driver does support the strong pullup (or it falls back to a pullup resistor). The DS18b20 temperature sensor specification lists a maximum current draw of 1.5mA and that a 5k pullup resistor is not sufficient. The strong pullup is designed to provide the additional current required.

The DS28EA00 provides an additional two pins for implementing a sequence detection algorithm. This feature allows you to determine the physical location of the chip in the 1-wire bus without needing pre-existing knowledge of the bus ordering. Support is provided through the sysfs w1_seq file. The file will contain a single line with an integer value representing the device index in the bus starting at 0.