1-Wire Interface on the OMAP

Matthew Percival

19th July 2005

Contents

1 The Usual Business

Any suggestions or errata would be most welcome, and shall be taken into immediate consideration.

2 Introduction

Being a little used bus, it is of no surprise that there is little documentation about the 1-Wire support on OMAP, and much of what is available is wrong. The hardware support does not work as advertised, and does not even properly comply to Dallas' 1-Wire specification, however with a little work it is possible to use 1-Wire devices. No OMAP boards come with any 1-Wire devices attached to date, but due to the simplistic nature of their design, it is easy enough to add one. All of my observations and testing have been on the OSK, however, much — if not all – of this should be accurate on other OMAP boards.

3 1-Wire Support on the OMAP

3.1 Getting Started

By mounting a 1-Wire device on the bottom of the board, you can begin to take advantage of the OMAP's 1-Wire support. Before you can begin to use the device, however, you need to do a little housework. Firstly, ensure the ARMXOR clock[1] is active — ARM_IDLCT2 [2] and ARM_RSTCT2 [0] should both be 1[2]. Next, you need to get the multiplexor settings right: HQD/1-Wire shares ball N20, which defaults to GPIO11 — FUNC_MUX_CTRL_6 [20:18] will need to be set to 1[3].

3.2 Registers

You can find the four registers associated with HDQ/1-Wire at base address 0xFFFB:C000[4]. All registers are 32bit access, however only the lowest 8bits are used in any of them. The registers are as follows:

HDQ1W_TX R/W, offset 0x00. You write data to send to the device here.

HDQ1W_RX R, offset 0x04. You read the data received from the device here.

HDQ1W_CTRL R/W, offset 0x08. Each field of this control register is used as follows:

SBM bit 7. Only used in 1-Wire mode, setting this bit enables single bit mode, whereby only bit 0 of the TX/RX registers is used.

IM bit 6. Setting this bit unmasks the device's three interrupts. Note that this only unmasks them at this level:

[1]Texas Instruments Inc., *OMAP5912 Multimedia Processor Clocks Reference Guide (spru751).*

[2]Texas Instruments Inc., *OMAP5912 Multimedia Processor OMAP 3.2 Subsystem Reference Guide (spru749).*

[3]Texas Instruments Inc., *OMAP5912 Multimedia Processor Pinout Reference Guide (spru769).*

[4]Texas Instruments Inc., *OMAP5912 Multimedia Processor Serial Interfaces Reference Guide (spru760).*

they still need to be correctly unmasked and configured in the Interrupt Controllers.

PDM bit 5. Setting this bit enables the clock to the bus.

GB bit 4. Setting this bit initiates a data transfer with the device. It clears itself after the transfer has been completed.

PD bit 3. Only used in 1-Wire mode, this read-only field is set if any devices responded to the reset pulse.

IP bit 2. Setting this bit sends a reset pulse along the bus. It clears itself after the pulse has been sent.

RWB bit 1. This bit determines whether the next action taken is to be a read or write operation: 1 is read, 0 is write.

MODE bit 0. This bit determines which mode to operate in: 0 is HDQ, 1 is 1-Wire.

HDQ1W_INTS R/C, offset 0x0C. A read to this register clears all interrupts. Each field is used as follows:

TC bit 2. Reports that a write to the device has completed.

RC bit 1. Reports that a read from the device has completed.

DTO bit 0. In 1-Wire mode, it reports that the reset pulse has completed; in HDQ mode, it indicates a read/write time-out.

3.3 Silicon Errata

Unfortunately, the control register (HDQ1W_CTRL) does not work as advertised. I have noted the following hardware bugs:

- The documentation[1] reads as if Single Bit Mode is persistent, however you need to set it with each read or write operation that needs to be in that mode. It is cleared immediately upon completion of each individual transfer.
- The PD bit is not set when 1-Wire devices respond to the reset pulse. This bit cannot be used to determine in advance if there are any 1-Wire devices on the bus.

[1] Texas Instruments Inc., *OMAP5912 Multimedia Processor Serial Interfaces Reference Guide (spru760)*.

- The reset pulse is not sent when you set the IP bit: you need to set GB too. Because IP is being treated the same as RWB, if the former is set when you are attempting to read or write, a reset pulse will be sent in place of your read/write request. Setting the GB a second time will ensure your read/write operation is undertaken.

- Though not dramatic, it should also be noted that the reserved bits of all HDQ/1-Wire registers always return 0: in the documentation, it claims they mirror bits 7:0.

A further hardware bug can be noted in the timings of 1-Wire operations: the OMAP support does not conform to Dallas' specification. The most significant issue is when the OMAP is sending or receiving a 1: the specification is to pull the signal low for 5-15μs at the start of the time slot, however the OMAP only does so for 1.3μs. This timing is compliant with the 1-Wire overdrive specification, accepting 1-2μs, however the OMAP has no option for switching between normal and overdrive modes. In addition, because not all 1-Wire devices support overdrive mode, some older devices will not work directly on the OMAP.

The reset pulse is also noteworthy in that the specification for normal mode is to pull the signal low for 500-640μs, which the OMAP just falls into at 500μs, however any signal over 480μs will take a device out of overdrive mode. This seems quite contradictory to the timing used for data transfer time slots, and as such the OMAP should be treated the same as 1-Wire adapters which do not support overdrive mode (such as the DS9097 serial adapter).

The official documentation[1] also reports that accessing the registers during a read/write operation can cause data corruption, and so it is recommended that interrupt driven I/O be used with all 1-Wire operations. This is the purpose of the omap_owire driver, as described in the next chapter.

Not a bug, but it should perhaps also be noted that you will be unable to write to some older 1-Wire EPROM devices with the OMAP, because they require 12V for a write operation. This is only

[1]Texas Instruments Inc., *OMAP5912 Multimedia Processor Serial Interfaces Reference Guide (spru760)*.

due to the boards using a lower voltage, rather than any hardware incapability of using the devices. Newer devices which use a lower voltage, however, are perfectly compatible with the OMAP boards.

3.4 Basic Operation

The following is based on the functional description in the official documentation[1], however it has some minor differences in where the actual operation differs from how it is advertised.

To use the device for 1-Wire operation, you first need to set the MODE bit in HDQ1W_CTRL to 1 (1-Wire Mode), and turn on the clock to the device by setting the PDM bit in the same register — these operations can be undertaken at the same time. Before even worrying about reading or writing, all operations in the 1-Wire protocol are initiated by sending a reset pulse to all devices on the bus. We do this by setting IP (send reset pulse) and GB (initiate transaction) in HDQ1W_CTRL: if there are any devices present, they will respond with a presence pulse.

Next, if you wish to write to the device, you need to follow these steps:

- Insert the byte to be written into the HDQ1W_TX register.
- Set RWB to 0 (write mode) and GB to 1 (initiate transfer) in the HDQ1W_CTRL register; if you wish to write only a single bit, also set SBM to 1 in the same register.
- After bits 7-0 (or just 0 in single bit mode) of HDQ1W_TX have been sent along the bus, the TC interrupt will be generated — at this point, the HDQ1W_INTS register should be read to clear the interrupt.

Finally, if you wish to read from the device, you need to follow similar steps:

- Set RWB to 1 (read mode) and GB to 1 (initiate transfer) in the HDQ1W_CTRL register; if you wish to request only a single bit, also set SBM to 1 in the same register.

[1] Texas Instruments Inc., *OMAP5912 Multimedia Processor Serial Interfaces Reference Guide (spru760)*.

- After bits 7-0 (or just 0 in single bit mode) have been received in the HDQ1W_RX, the RC interrupt will be generated — at this point, the HDQ1W_INTS register should be read to clear the interrupt.
- Reading the HDQ1W_RX now will provide the data send from the device.

4 The omap_owire Kernel Module

Because interrupt driven I/O is recommended for the OMAP hardware support[1], I wrote a Linux driver for this purpose: that is omap_owire. This document describes the 1.0 version of the driver, both in how it works and how to use it. It has been written for Linux-2.6.12-omap1, and with an extended mux.h: if your kernel was not compiled with these extended mux entries, or if your kernel version is significantly different to this one, then the driver will not work.

4.1 Module Workings

Most of the basic aspects of the module are fairly self-explanatory, but I shall explain what several aspects do, and why I have included them. I shall also comment on any future changes I am planning on making. This driver currently makes trivial use of the procfs, however, once I learn how to properly make use of sysfs, I will likely remove the procfs-related code, and replace it with far more useful sysfs files. I am also considering buffering the read/write data in kernel space between the user and registers: this would mean that read and writes may have to be broken up into smaller blocks — either on kernel- or user-side — however it would make room for smarter code.

4.1.1 omap_owire_status

The omap_owire_status struct acts as a basic state machine: it allows the module to keep track of what it is currently doing. It has the following members:

opened which lets the module know if the device is already open: only one program may use the device at a time.

detecting after being set to true, the module polls this variable — when it returns to false, the DTO interrupt has been received.

reading after being set to true, the module polls this variable — when it returns to false, the RC interrupt has been received.

[1] Texas Instruments Inc., *OMAP5912 Multimedia Processor Serial Interfaces Reference Guide (spru760)*.

writing after being set to true, the module polls this variable — when it returns to false, the TC interrupt has been received.

operating lets the module know that it is in the middle of a command sequence. If not set, the next write is interpreted as a command character.

searching is used to note that the module is currently undertaking a Search ROM sequence — this is the only instance where Single Bit Mode is required. When searching reaches 64, the module knows it has completed the Search ROM sequence, and returns to byte transfers[1].

4.1.2 omap_owire_init()

This function is called when the module is loaded, and sets the system up for 1-Wire operation. It begins by registering the owire device with major number 232 — if this fails, it returns -EBUSY. The next step is the multiplexed ball N20, which is set to HDQ mode. The device then tries to acquire the ARMXOR clock: in the unlikely event this fails, it will return the appropriate error. The module finally sets the HDQ/1-Wire system to 1-Wire Mode. If procfs is configured in the user's kernel, the module also creates /proc/owire, returning -ENOMEM upon failure.

4.1.3 omap_owire_ctrl_proc_read()

If procfs were compiled in kernel, this is the function called when a user reads /proc/owire. The function is fairly simple: it merely loads the HDQ1W_CTRL register and returns a string output based on its current values. Because I am likely to move this device to sysfs, there are no plans to expand on this; when the device does make use of sysfs, the replacement functions should have greater functionality.

4.1.4 omap_owire_open()

Called when a user-space program opens the owire device, this function undertakes the final steps in preparing the bus for use. Firstly, if the bus has already been opened by another user, the

[1] Dallas Semiconductor, *1-Wire Search Algorithm (app187)*.

function will return -EBUSY — only one program may use the bus at a time, so as to prevent multiple programs from interfering with each other.

If the bus is available, the clock is turned on and interrupts are configured and unmasked; the omap_owire_interrupt() interrupt service routine, which is added to the vector table here, merely clears the interrupt and resets the appropriate state in omap_owire_status. omap_owire_fsync() is called, to ensure that the states recorded in omap_owire_status are reset, followed by a reset pulse being sent. The function then sends a 0xF0 (Search ROM command) to the bus and reads back the value. A 0 indicates a short circuit — the function returns -ESTRPIPE — and a 0xF0 indicates there are no devices present — the function returns -ENODEV.

Assuming all has worked fine, the function locks the bus from being opened by any further programs, while also obtaining a lock on the module.

Though most of this function is fairly set now, returning -ESTRPIPE on a short circuit was a case of 'close enough.' I may change that to a more appropriate error code in future, should I identify something that is more applicable to this situation.

4.1.5 omap_owire_read()

Called when a user-space program reads from the 1-Wire bus, this function begins by checking the the module's current state. If the module is not currently operating under any mode, it calls owire_read_byte() to fill the buffer with the number of bytes specified by the user — according to the 1-Wire protocol, these reads will always be 0xFF however, and as such are a pointless operation. In future a more intelligent manner of handling this case will be considered.

If the module is currently operating, then it checks if it is currently following the Search ROM algorithm. If so, then it calls owire_read_bit(), otherwise it calls owire_read_byte(). The Search ROM algorithm requires reads of two bits, while reads are normally of at least one byte. Due to the fact that these reads are always in pairs, I will consider enforcing this rule: currently the

user can request more, which would cause them to break from the Search ROM algorithm, and need to restart the process[1].

For reference, owire_read_byte() will read a specified number of bytes from the 1-Wire bus, and return them to the user-specified buffer. wire_read_bit() works the same, however it operates in Single Bit Mode. If the function exits normally, it returns a bit/byte count of how much it has read.

4.1.6 omap_owire_write()

Called when a user-space program writes to the 1-Wire bus, this function begins by checking the module's current state. If the module us not currently operating under any mode, it assumes the next byte is a command byte and checks what it is. If the next byte is 0xF0, then the module identifies that the user is about to enter Search ROM mode and sets the appropriate state in omap_owire_status. However, if the nest byte is either 0x3C or 0x69, the user is requesting overdrive modes not supported by the OMAP, so the function returns -EPROTONOSUPPORT. Otherwise, the operating state is set in omap_owire_status and owire_write_byte() is called.

If the module is currently operating, then it checks if it is currently following the Search ROM algorithm. Is so, then it calls owire_write_bit(), otherwise it calls owire_write_byte(). The Search ROM algorithm requires writes of a single bit, while writes are normally of at least one byte. Due to the fact that these writes are always singular, I will consider enforcing this rule: currently the user can send more, which may cause them to break from the Search ROM algorithm, and need to restart the process[1].

For reference, owire_write_byte() will write a specified number of bytes from the user-specified buffer to the 1-Wire bus. wire_write_bit() works the same, however it operates in Single Bit Mode. If the function exits normally, it returns a bit/byte count of how much it has written.

[1]Dallas Semiconductor, *1-Wire Search Algorithm (app187)*.
[1]Dallas Semiconductor, *1-Wire Search Algorithm (app187)*.

4.1.7 omap_owire_fsync()

Before I describe the function, I will start by making it clear that I am aware this is not a correct use of the fsync function. I had originally wanted to use flush(), as that is much closer to the actual purpose of this function, but because that would not have been available to my user-space C programs, I ended up choosing fsync() in its place. Should flush() become available in future, I will correct this immediately

Called when the user-space program makes a fsync request, this function simply resets the state machine in the module — with the exception of the 'opened' state — and sends a reset pulse along the bus.

4.1.8 omap_owire_release()

Called when a user-space program closes the owire device, this function simply undoes what was done by omap_owire_open(). It masks and releases the interrupt, turns the clock off to the bus, and releases the locks on the device and module.

4.1.9 omap_owire_exit()

Called when the module is unloaded, this function has a fairly simple task. The ARMXOR clock is released, and the device unregistered. Ball N20 is restored to GPIO mode, as is the default configuration. If it were created, /proc/owire is removed.

4.2 Module Use

Before using the driver, you will need to make a device file with major number 232 — because this number has not been officially assigned to 1-Wire, it may be forced to change in future. You can create this device like so:

```
# mknod /dev/owire c 232 0
```

To use the module, you will need to ensure you have the omap_owire.ko file in your /lib/module/${KERNEL_VERSION}/ directory, and have the following line in your modules.dep:

/lib/modules/${KERNEL_VERSION}/omap_owire.ko:

Once you have done so, you can load the module with modprobe:

```
$ modprobe omap_owire
Using /lib/modules/2.6.12-omap1/omap_owire.ko
1-Wire for TI OMAP.
MUX: initialized N20_HDQ
```

Further operation is fairly simple with the following C functions:

open() If the device is not currently in use, this will give your program exclusive access to it.

read() Allows you to read a specified number of bytes from the device, which will be placed in the specified buffer.

write() Allows you to write a specified number of bytes to the device from the buffer specified; in a Search ROM situation, only the first bit of each byte will be used.

fsync() Allows you to reset the bus: this is required by the 1-Wire protocol before sending any command bytes.

close() When your program has finished with the device, it must call close(): if you do not, no other programs will be able to use it, nor will you be able to unload the module.

Below is a simple program demonstrating the use of these functions:

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>

int main(void)
{
int fd;
char command = 0x33;
char serial[8];

/* open the 1-Wire device */
```

```
if ((fd = open("/dev/owire", O_RDWR)) < 0)
{
fprintf(stderr, "ERROR: cannot open /dev/owire\n");
exit(1);
}

/* send a Read ROM command to the device */
if (write(fd, &command, sizeof(char)) < 0)
{
fprintf(stderr, "ERROR: cannot write to /dev/owire\n");
exit(1);
}

/* read the ROM from the device */
if (read(fd, serial, 8*sizeof(unsigned char))) < 0)
{
fprintf(stderr, "ERROR: cannot read from DSP\n");
exit(1);
}

printf("The device's family code is: %#X\n", serial[0]);
printf("The device's serial code is: %#X%X%X", serial[1],
    serial[2], serial[3]);
printf("%X%X%X\n", serial[4], serial[5], serial[6]);
printf("The device's CRC is: %#X\n", serial[7]);

/* reset the bus */
fsync(fd);

/* close the device */
close(fd);

return 0;
}
```

You can also view the current values of each field in the HDQ-1W_CTRL register at any time by reading the /proc/owire file, for example:

```
$ cat /proc/owire
MODE: 1-Wire Mode
RWB: Write Mode
IP: Not Currently Polling
PD: Field Not Presently Working
GB: Device Idle
PDM: Clock Disabled
IM: Interrupts Disabled
SBM: Single Bit Mode Disabled
```

5 OMAP Support in owlib

This is almost a footnote, really, but worth including for the sake of completeness. Having OMAP 1-Wire support in a kernel module is all well and good, but you then need a user-space program to be able to do anything useful with the device. Due to the large number of different 1-Wire devices available, this could be painful if you are using many different devices. To ease this, I have also recently written in support for omap_owire into owlib.

owlib is a major part of the owfs project (http://owfs.sourceforge.net/), and allows you to access 1-Wire devices in a variety of different, simple ways. Currently, the project allows a filesystem interface, a web interface, and bindings for Perl, Python, PHP and Tcl. It is not too difficult to add support for any 1-Wire devices they have missed, and with OMAP support now operable, this is a simple option available.

6 Bibliography

Below are a few documents which may be useful in coming to a better understanding of various topics raised in this document.

Dallas Semiconductor, '1-Wire Search Algorithm', <http://pdfserv.maxim-ic.com/en/an/app187.pdf>, (accessed 13/07/05).

Texas Instruments Inc., 'OMAP5912 Multimedia Processor Clocks Reference Guide', <http://www-s.ti.com/sc/psheets/spru751a/spru751a.pdf>, (accessed 05/07/05).

Texas Instruments Inc., 'OMAP5912 Multimedia Processor OMAP 3.2 Subsystem Reference Guide', <http://www-s.ti.com/sc/psheets/spru749a/spru749a.pdf>, (accessed 15/06/05).

Texas Instruments Inc., 'OMAP5912 Multimedia Processor Pinout Reference Guide', <http://www-s.ti.com/sc/psheets/spru769a/spru769a.pdf>, (accessed 16/05/05).

Texas Instruments Inc., 'OMAP5912 Multimedia Processor Serial Interfaces Reference Guide', <http://www-s.ti.com/sc/psheets/spru760b/spru760b.pdf>, (accessed 04/07/05).