Linux I2C slave testunit backend

by Wolfram Sang <wsa@sang-engineering.com> in 2020

This backend can be used to trigger test cases for I2C bus masters which require a remote device with certain capabilities (and which are usually not so easy to obtain). Examples include multi-master testing, and SMBus Host Notify testing. For some tests, the I2C slave controller must be able to switch between master and slave mode because it needs to send data, too.

Note that this is a device for testing and debugging. It should not be enabled in a production build. And while there is some versioning and we try hard to keep backward compatibility, there is no stable ABI guaranteed!

Instantiating the device is regular. Example for bus 0, address 0x30:

```
# echo "slave-testunit 0x1030" > /sys/bus/i2c/devices/i2c-0/new_device
```

After that, you will have a write-only device listening. Reads will just return an 8-bit version number of the testunit. When writing, the device consists of 4 8-bit registers and, except for some "partial" commands, all registers must be written to start a testcase, i.e. you usually write 4 bytes to the device. The registers are:

0x00 CMD - which test to trigger 0x01 DATAL - configuration byte 1 for the test 0x02 DATAH - configuration byte 2 for the test 0x03 DELAY - delay in n * 10ms until test is started

Using 'i2cset' from the i2c-tools package, the generic command looks like:

```
# i2cset -y <bus_num> <testunit_address> <CMD> <DATAL> <DATAH> <DELAY> i
```

DELAY is a generic parameter which will delay the execution of the test in CMD. While a command is running (including the delay), new commands will not be acknowledged. You need to wait until the old one is completed.

The commands are described in the following section. An invalid command will result in the transfer not being acknowledged.

Commands

0x00 NOOP (reserved for future use)

0x01 READ_BYTES (also needs master mode)
    DATAL - address to read data from (lower 7 bits, highest bit currently unused) DATAH - number of bytes to read

This is useful to test if your bus master driver is handling multi-master correctly. You can trigger the testunit to read bytes from another device on the bus. If the bus master under test also wants to access the bus at the same time, the bus will be busy. Example to read 128 bytes from device 0x50 after 50ms of delay:

```
# i2cset -y 0 0x30 0x01 0x50 0x80 0x05 i
```

0x02 SMBUS_HOST_NOTIFY (also needs master mode)
    DATAL - low byte of the status word to send DATAH - high byte of the status word to send

This test will send an SMBUS_HOST_NOTIFY message to the host. Note that the status word is currently ignored in the Linux Kernel. Example to send a notification after 10ms:

```
# i2cset -y 0 0x30 0x02 0x42 0x64 0x01 i
```

0x03 SMBUS_BLOCK_PROC_CALL (partial command)
    DATAL - must be '1', i.e. one further byte will be written DATAH - number of bytes to be sent back DELAY - not applicable, partial command!

This test will respond to a block process call as defined by the SMBus specification. The one data byte written specifies how many bytes will be sent back in the following read transfer. Note that in this read transfer, the testunit will prefix the length of the bytes to follow. So, if your host bus driver emulates SMBus calls like the majority does, it needs to support the I2C_M_RECV_LEN flag of an i2c_msg. This is a good testcase for it. The returned data consists of the length first, and then of an array of bytes from length-1 to 0. Here is an example which emulates i2c_smbus_block_process_call() using i2ctransfer (you need i2c-tools v4.2 or later):

```
# i2ctransfer -y 0 w3@0x30 0x03 0x01 0x10 r? 0x10 0x0f 0x0e 0x0d 0x0c 0x0b 0x0a 0x09 0x08 0x07 0x06 0x05 0x04 0x03 0x02 0x01 0x00
```