One-page report on possible extensions of the driver.

# Accept calls from multiple user threads

The driver is operated in kernel space and always be accessed by any number of user threads. When accessed by multiple user threads the problem is Race condition.

*To avoid race condition*

1. *The global data/shared data need to be protected:*
2. Some global data need to be made thread local data.
3. Concurrent writes has to serialize.

**Global/shared data: (Current page pointer)**

The EEPROM has one page pointer to maintain the page address for read and write operations. If multiple threads are accessing the driver there needs to be protection and synchronization between all threads for the page pointer location.

In the existing code the page pointer is local to client. But the device is common to all threads and the page pointer inside the device is shared. Hence one global page pointer is to be declared and protected by the semaphore.

**Thread local data: (work queue structures)**

When a read/write request is received the user buffer is copied into work structure and then processed. The work structures has to be thread local. The current driver keeps them as global variables and operates. If they are global when accessed concurrently the thread context is lost. This leads to misinterpretation. If Thread1 issues the read request and thread also issues read request, the buffer of thread 2 overwrites the buffer of thread1 hence the thread gets the thread1 request.

The solution for above problem is to move the work structures to device structure which resides in file\_private\_data which is thread local. Hence when the two threads made the request their requests are kept to their own local structures hence the misinterpretation is resolved.

**Concurrent writes serialization**:

When multiple threads issue the write the driver has to write into one device. Hence there should be a mechanism to serialize all the write requests. Hence a global write request semaphore has to be implemented. Whoever is calling the write function (i2c\_master\_send) has to hold the lock until the job is completed. This comes with a price to block the other threads while one of them is using. To ensure proper write safety this price has to be paid.

# To work on different EEPROM chip

The Existing driver has defined macros for the device specific properties like Slave address, Page count, Page size and Bus number. Re numbering the macros manually and recompiling the driver will work for different device. The driver can be added some more conditional macros to detect the type of device and load the required properties into macros will make the driver work all kind of device at the runtime.