**SAMSUNG OFA DRIVER Release 1.4 PLANNING MEETING WORK ITEM REQUEST LIST**

1. **NVMeResetBus routine:**

**a) don’t need StorPortSynchronizeAccess since IO** completions not done in ISR context anymore (as was done when COMPLETE\_IN\_DPC flag existed). However, still must not schedule a DPC from this routine (as was originally done) as all work must be completed before returning.

Recommendation: Call **NVMeSynchronizeReset()** function directly from this routine and do not return from it till all work is completed. Also, maybe rename **NVMeSynchronizeReset** as there is no outstanding synchronization issue.

b) **NVMeSynchronizeReset()** routine doesn’twait for all necessary work to be done before returning. It falls straight through after calling NVMeRunningStartAttempt to restart initialization state machine without waiting for it to complete. It allows IOs to resume and returns to NVMeResetBus which, in turn, also returns.

Solution: Wait for the initialization state machine to run to completion in NVMeSynchronizeReset before allowing IOs to resume and returning. Specifically, add logic similar to that in NVMePassiveInitialize, which waits for pAE->DriverState.NextDriverState to become either NVMeStartComplete or NVMeStartFailed in a while loop which calls NVMeStallExecution between checks, up to some maximum amount of time.

2. The **Recovery DPC** routine has the same problem NVMeSynchronizeReset above – there is no logic in place to wait for the initialization state machine to run to completion after the call to NVMeRunningStartAttempt() which starts it off.

3. **NVMeWaitForCtrlRDY should have a return value of type BOOLEAN that can be checked to see if it was successful or not.** Currently, wherever this is called, the driver forges ahead regardless of whether the RDY bit actually is in the desired state or not.

4. **NVMeCompleteCmd should have a return value that can be checked to see if it was successful or not.** Right now, wherever it’is called from the code forges ahead regardless of whether it succeeded or failed:

VOID NVMeCompleteCmd{

. . .

if ((pCmdEntry->Pending == FALSE) || (pCmdEntry->Context == NULL)) {

/\* Something bad happened so reset the adapter and hope for the best \*/

                  NVMeResetController(pAE, NULL);

                                    return;

}

As shown above, one of the routines which NVMeCompleteCmd calls is NVMeResetController. Since NVMeCompleteCmd has no return value, **this fatal error return is never detected** in any of the places that the function is called from (quite a few) - the logic just proceeds on as if everything is fine. In some cases NVMeCompleteCmd can be called over and over (if it is called from DetectPendingCmds or IoCompletionDpcRoutine for example) which may in turn cause repeated calls to NVMeResetController.

5. **There is redundancy in the new routine NVMeWaitForCtrlRDY() and the old routine NVMeWaitOnReady()**. We don’t need both – we can get rid of the old routine (though as noted in item #3 above, the new routine needs a return value).

6. In NvmeStd.c, line 646:

**Except for the first sentence, this comment is not accurate,should be removed:**

/\*

\* Before we transition to 0, make sure the ctrl is actually RDY

~~\* NOTE:  Some HW implementations may not require this wait and  if not then it could be removed as waiting at this IRQL is  not recommended.  The spec is not clear on whether we~~

~~\* need  to wait for RDY to transition EN back to 0 or not.~~

\*/

NVM Express 1.0e and beyond includes the following statement in the definition of the EN bit(emphasis added): “Setting this field from a ‘0’ to a ‘1’ when CSTS.RDY is a ‘1,’ or setting this field from a '1' to a '0' when CSTS.RDY is a '0,' has undefined results.”

7. The routine **NVMeResetAdapter()** sets CC.EN to 0 without ever checking to make sure that CSTS.RDY is set to ‘1’ first. This check has to be included in this routine. Since it is not, there are many paths in the driver where there is no prior check for this condition:

               a) NVMeInitAdminQueues -> NVMeEnableAdapter -> NVMeResetAdapter

b) NVMeNormalShutdown -> NVMeResetAdapter

c) NVMeAdapterControlPowerDown -> NVMeResetAdapter

d) NVMeSynchronizeReset -> NVMeResetAdapter

8. In the **RecoveryDpcRoutine()**:

a) the code does not need to set CC.EN to ‘0’ and then wait for CSTS.RDY to become 0 because right after it does so, it calls NVMeResetAdapter which does the exact same thing.

b) is it really safe and/or required to always acquire/release the StartIo lock?

9. **StartIo spinlock deadlock** - when the flag to indicate multiple cores are sharing a single queue, **MultipleCoresToSingleQueueFlag is set, we’ve seen a deadlock situation** for some commands.

Example Scenario (Start Stop Unit command):

In the IoCompletionDpcRoutine(),theStartIoLock is taken.

IoCompletionDpcRoutine()->SntiCompletionCallbackRoutine()->SntiTranslateStartStopUnitResponse(), ProcessIo() is invoked with the AcquireLock parameter set to TRUE.

In ProcessIo(), when AcquireLock is set, driver tries to take the StartIoLock spinlock. Hence deadlock.

We’ve implemented mode sense in similar fashion and seen the same issue. We have uncommented the log sense command and seen the same issue in SntiTranslateTemperatureResponse().

10. **WHCK test failure** - when IoStatus is set to NOT\_SUBMITTED in ProcessIo(), the SRB is not completed. This causes WHCK test case "DP WLK-Hot-Add-Device Test -Verify driver support for Hot -Add CPU" to crash.

 Solution: add the following code at end of ProcessIo():

if (IoStatus == NOT\_SUBMITTED) {

if (pSrbExtension->pSrb != NULL) {

pSrbExtension->pSrb->SrbStatus = SRB\_STATUS\_ERROR;

IO\_StorPortNotification(RequestComplete, pAdapterExtension, pSrbExtension->pSrb);

     }

}

11. During **Passive initialization, the timeout needs to be increased** as the driver initialization takes more time in Server PC’s compared to Client PC’s.

12. **NVMeAdapterControl** miniport routine - sometimes returns illegal value. WDK specifies that the driver must always return ScsiAdapterControlSuccess. However, depending on execution, the driver may currently return ScsiAdapterControlUnsuccessful for ScsiStopAdapter and ScsiRestartAdapter control types.

13. New Bus Type **NVME - support for new “NVME” Bus Type.** This is supported in WDK 8.1 / Visual Studio 2013. Note: this will result in seven extra WHQL test cases.

14. **Memory corruption** constructing inquiry response data - In SntiTranslateStandardInquiryPage(), the following line of code is touching a field way past the end of STANDARD\_INQUIRY\_LENGTH (36 bytes):

pStdInquiry->Reserved3[0] = RESERVED\_FIELD;

15. **Add support for Read Only mode** – to support Read Only Volumes, add support for RO mode as per discussions in NVMe technical committee/technical reflector.

16. **WHQL SCSI Compliance issues** - following are some changes the driver needs to make in order to pass WHQL SCSI Compliance test:

 a) Add support for SCSI Log Sense command

 b) Implement support for EUI64 for SCSI inquiry command when translating unit serial page and device identification page

 c) Implement checking for ServiceAction field in SntiTranslateReadCapacity16.

 d) When processing SCSI request sense command, using Sense Data, sense buffer should be used rather than data buffer.

 e) MODE\_CACHING\_PAGE\_EX , rather than MODE\_CACHING\_PAGE, needs to be used for 2012 and 2012R2